REPORT OF DISSERTATION WORK

ON

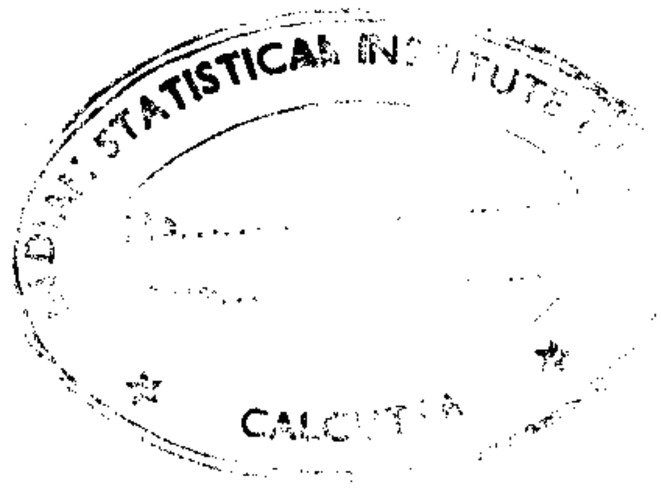
DESIGN OF A MOBILE AGENT SOFTWARE FOR APPLICATION IN DATAMINING

BY K. RAJESH BABU **(MTC9620)**

Under the supervision of

PROF. ADITYA BAGCHI

Mr. SUBHAMOY MAITRA



INDIAN STATISTICAL INSTITUTE , CALCUTTA

YEAR 1997 - '98

Acknowledgements

I would like to express my sincere thanks to my Guides **Dr. Aditya Bagchi** and **Mr. Subhamoy Maitra** for suggesting me to work in this interesting field. I am grateful for all the suggestions and guidance they provided throughout the work. Thanks to my batch mates who also provided help in one form or other.

Certificate of Approval

This is to certify that the dissertation work entitled **DESIGN OF A MOBILE AGENT SOFTWARE FOR APPLICATION IN DATAMINING** submitted by **Rajesh** Babu, in partial fulfilment of the requirements for M.Tech. in *Computer Science* degree of the *Indian Statistical Institute*, *Calcutta*, is an acceptable work for the award of the degree.

Date: July 28, 1998.

(Supervisor)

(External Examiner)

Index

INTRODUCTION

1.1 What are Agents?

Agents in general are entities who represent a person and carries out certain jobs instead of him. Software Agents are pieces of code, that are Autonomous, that have Social ability, that are Reactive and Proactive.

Because of Autonomy, the Agents does not need any intervention from human user and they have some control on their own state. Social ability makes the agents collaborate with other agents or humans and achieve greater tasks which would have been impossible for an agent alone, for example an agent who wants to search databases, but is good at presenting information to human user, can collaborate with another agent which is good at searching databases. Agents sense their environment and respond accordingly, for example arrival of an email can trigger an agent to take particular action. Agents not only react to changes in the environment, they also initiate certain actions, exhibiting goal-driven behaviour, for example an agent which is supposed to fix up an appointment with some person may contact that person by some form of communication.

In everyday life, one has used one form of agent or other. For example, mail reader or news reader applications are agents in a way, they are customised with server name and when to fetch mail and in what way to sort the mail/news items. Once it is customised, these applications at proper time fetch the mail/news and sort them in the specified order. Certain applications even tries to find out which mails may be urgent ones by searching for words like 'urgent' or 'emergency' etc. and put those mails on top of the mailbox.

Agents fit in the present day network centric world well. For example, if a person wants to search information about a topic of particular interest, the search facilities provided by present generation search engines produce voluminous amount of information, from which required information is to be filtered. If the network bandwidth is very low and the connectivity is also not proper, for example, in case of a home user, then the person has to spend a lot of time before the terminal, waiting for the information to load in his browser and then sift through them and refine the search on and on, till he finds what he needs. If in between the network connection is broken then the search is to be carried out from the start, unless previous search results were stored locally. Agent with the logic for filtering the information needed, embedded in it can download the information when the connectivity is good and off-line can search through them and report the final filtered result, thus freeing its user from this tiresome work.

Agent can also use existing Artificial Intelligence techniques to imitate human beings or playing as a good interface between the computer and normal user. Agents can be classified based on whether they are intelligent or not, whether they are mobile or not, the kind of expertise they possess.

1.2 Mobile Agents

Stationary agents execute tasks on processor, but can spawn other tasks on the same processor or other processors(on other machines). Mail/News readers and applications which react to important emails and applications which helps in searching the Internet are some examples of stationary agents, they are generally bulky and also monolithic. But in this network centric world, where network resources are made available to everyone through the Internet, it is possible to have agents which can move from one machine to another carrying along with it its code and its state. These agents, which can move from one machine to another carrying along with it its state and code and continue executing on the destination machine, are termed Mobile Agent. For example, we can have an Mobile agent to visit other machines and deliver some important messages to various people, instead of sending it by email.

The concept of Mobile agents arose out of developments in Distributed Computing and Distributed Artificial Intelligence. In Distributed computing paradigm, the central principle of computing followed was RPC(Remote Procedure Call), enabling one computer to make calls into procedures available in a remote machine.

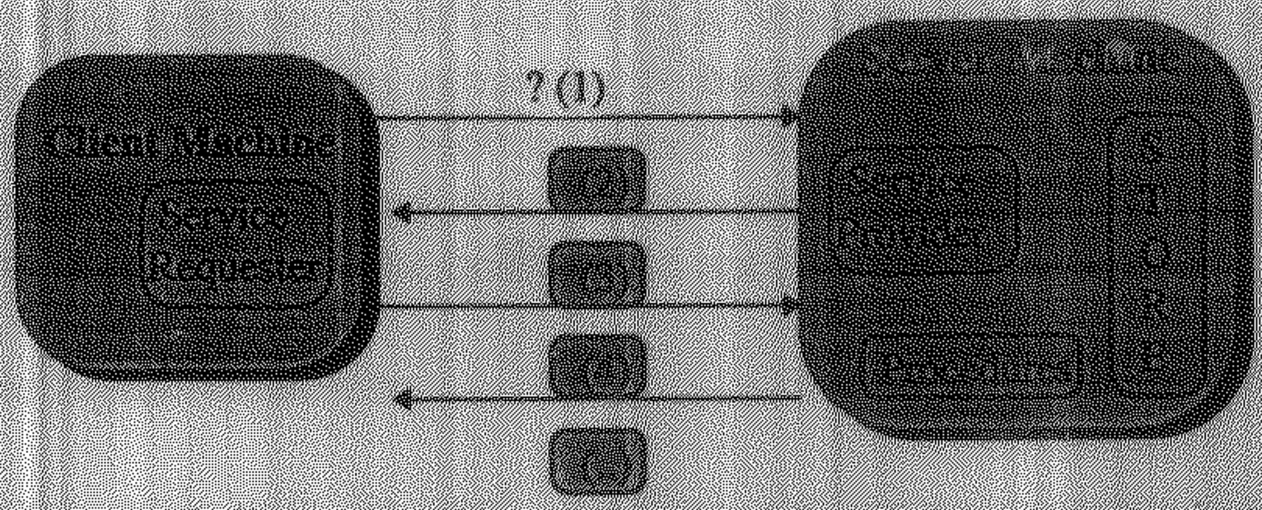


Fig 1.1 RPC mechanism

The above diagram shows how one computer(client) makes calls into procedures present in another machine(server). The client sends a request for particular procedure call along with the parameters needed. The server side makes the procedure calls, and the result is returned to the client. The computers which follow RPC paradigm agree in advance the procedures provided and the parameters to be passed as arguments and their type etc.

A complicated task may need a series of voluminous amount of request followed by equal number of replies, hence if the connectivity is less and the network bandwidth is less, then results don't persist across connectivity losses and traffic is high. The client can not extend the set of procedures given at the server end, the additional logic is provided only on client machine.

In another paradigm known as Remote Programming (RP) or Mobile Agent computing, the client sends the code to the server machine, and the code sent there makes use of the API calls provided by the server and executes locally and gets the result and reports back to the client. This is illustrated in the following diagram.

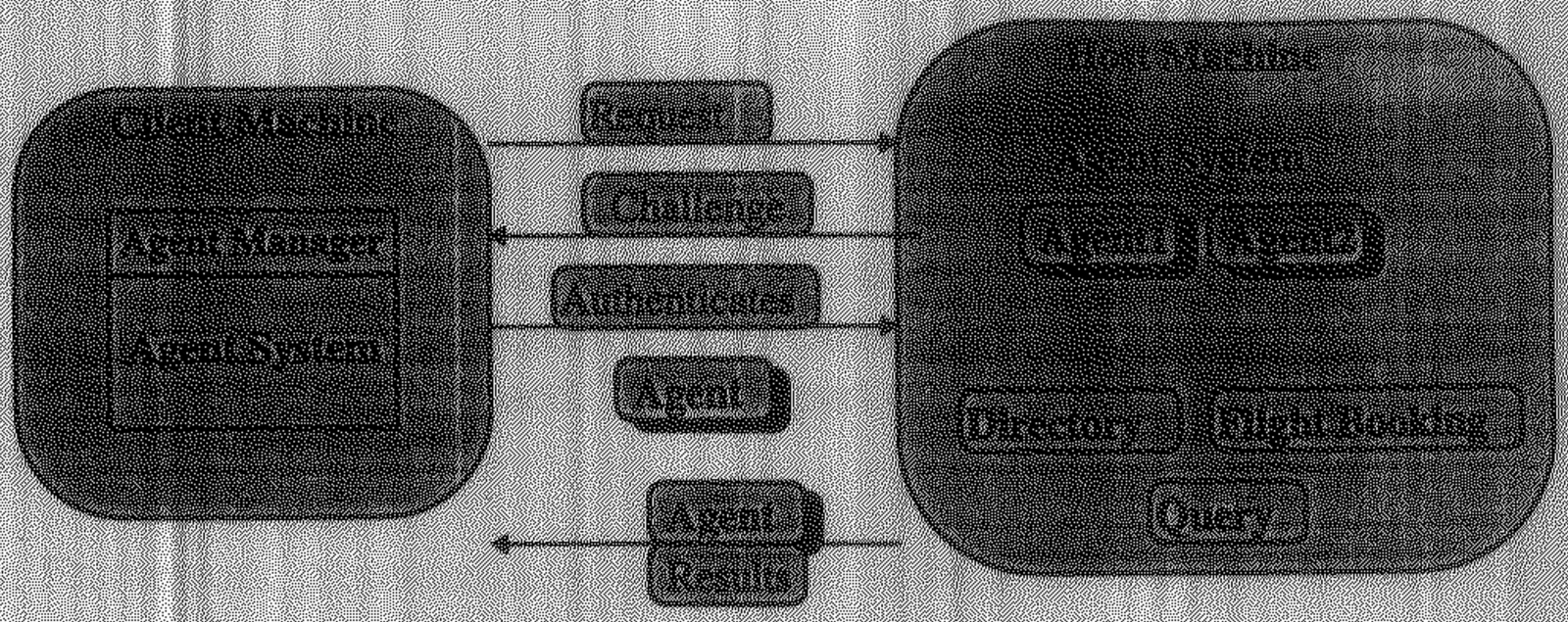


Fig 1.2 RP

This paradigm(RP) offers advantage over RPC, if the network connectivity is not reliable and the bandwidth is less. If the overhead for transferring the agent is less, as is in the case of home user, compared to requests and results shuttling back and forth, then once the agent is transferred the agent executes locally, hence network resources are not wasted. Generally RP paradigm supports Object-Oriented approach, hence the agent can extend the functionality provided by the server, which is not possible in case of RPC.

But since some unknown code is allowed to visit one's machine or after sending it to some hosting machine and retracting it back, lot of security problems arise. The Framework which supports Mobile Agents must have sufficient security built-in to protect the host machine and also protect one agent form another agent. The agent also needs the privacy, i.e. the host should not tamper with the agent's code.

1.3 Mobile Agent Concepts

Many agents each with its goal specified and communicating with each other to achieve the goal, is how distributed applications are to be constructed in RP. Agents move from one machine to another carrying its state and code. If Agent needs lot many custom objects not present in a standard hosting system, and if it carries all of them with it, then transferring overhead increases, but once transferred it can execute with out any hitch.

Mobile agent technology also gives lot of freedom to Distributed System designer, since functionality can be shifted either to client or server side. The designer does not have to worry about performance issues to decide which side should have a particular functionality. The functionality can be shifted dynamically to the place where work can be done at efficient manner.

Places are contexts where agents execute. Many places constitute an **Agent System**, each Agent System is associated with an **Authority**, whom it represents. Different Agent Systems of same authority are grouped together to form a **Region**. An agent Host machine will contain many regions. All host systems provide some sort of Directory service to find out what are all the services provided.

1.4 Contribution

In this report, a survey of some of the Existing Mobile Agent Systems is presented. The Architecture of KOSAgent(Kinetic Object Space Agent) JAF, a new Java Based Agent Framework, is proposed. Then the part of the JAF which has been implemented is explained in detail.

The Framework presented here is designed to be a generic one. Because of want of time, the application to datamining has not been implemented.

Existing Mobile Agent Systems

2.1 Existing Systems

Most important issues in constructing a system for hosting mobile agents, is to protect an agent host from hostile agent and vice-versa, make agent transfer transparent to the programmer and make code transfer efficient. If an agent system is deployed for commercial purpose, for example providing mobile agent based booking service, then secure transactions must be made possible. The Client who sends his agent to some agent system, should have assurance that his private information will remain private. As operating systems does not provide support for mobile code, the solution is to use a VM which runs on the OS and provides the necessary services(Mobility, Directory, Query and other specific services).

An agents state can be described by[9] : 1) the code(or ***program state***) 2) the contents of instance variables(or ***data state***) 3) ***execution state***(local variables, threads etc.). A system which transports all these states to another destination, are said to use ***strong migration***. A system which transports program state and data state only is termed to use ***weak migration***.

Among the many existing agent systems, not many systems are ready to be deployed on the Internet(for commercial purpose), still much research is being carried out. In this section, Telescript agent system, Aglets[5], AgentTcl[7], ARA[16], Kasbah[15], Mole agent system - some of the existing agent systems developed by Research Institutes and Software Companies are briefly discussed.

One important feature of most of the existing agent systems is that, they all use some form of interpreted language. Another feature is that most systems are Object-Oriented, hence existing agents can be extended to create customised agent for the specific needs, providing reuse and reduced development cost. The languages generally used are Java, SmallTalk, Tcl, Python etc. Since compiled code gives better execution speeds, the stationary parts of some agent systems are native applications.

Some of the projects being carried out are trying to use popular languages like C/C++, for developing Agent Systems. Interpreters for these languages have been designed with eye on achieving native code speed.

Some of the systems have been developed with targets as Enterprises, providing centralised control over the agent system. Also certain systems encrypt the agent state information before transporting them.

Since this is a new field, in which no standard has been established(effort is underway) and each implementer has his own architecture and facilities provided in his system. The promising use of agents, Electronic Commerce, needs a global framework, so that the clients and service providers should have choice of the Agent framework to use and agents created for one framework should be able to execute in another framework. In order to make this interoperability a reality, work is going on.

Certain agent systems provide agent services, but external agents are not allowed inside or agents created in the system are not mobile(implying that they don't visit travel to other systems). Even though this is not a mobile agent system, this system provides services to agents, and agents are created upon requests from the clients and customised for various activities specified by the client, the agents execute in the agent system interacting with other agents , using the local services and resources. The client can later contact the system, specifying the agents ID which was given when the specific agent was created and inquiring about the status and results are reported by the agent. One such implementation is Kasbah, the electronic market place for agents for selling and buying goods, this system uses Web interface for the clients to request for service through HTML Forms, when the Form is submitted Agents(Processes) are spawned which share the processor with other Agents(Processes) on the Server and later submit the results when the client asks for.

Another such example is the AgentSoft's LiveAgentPro[10], which also provides a Web based interface with Java Applet and agent can be customised and the Applet sends the agent back to the server where it executes to complete the assigned duty. Some example agents (like Book Finder Agent)are available at their web site.

CORBA(Common Object Request Broker Architecture), is the Industry standard middle-ware a product of a consortium called OMG[13](Object Managenment Group). It is the specification for emerging technology DOM(Distributed Object Management). DOM technology provides a higher-level, object-oriented interface on top of the basic distributed computing services. The next figure shows the Object Management Architecture.

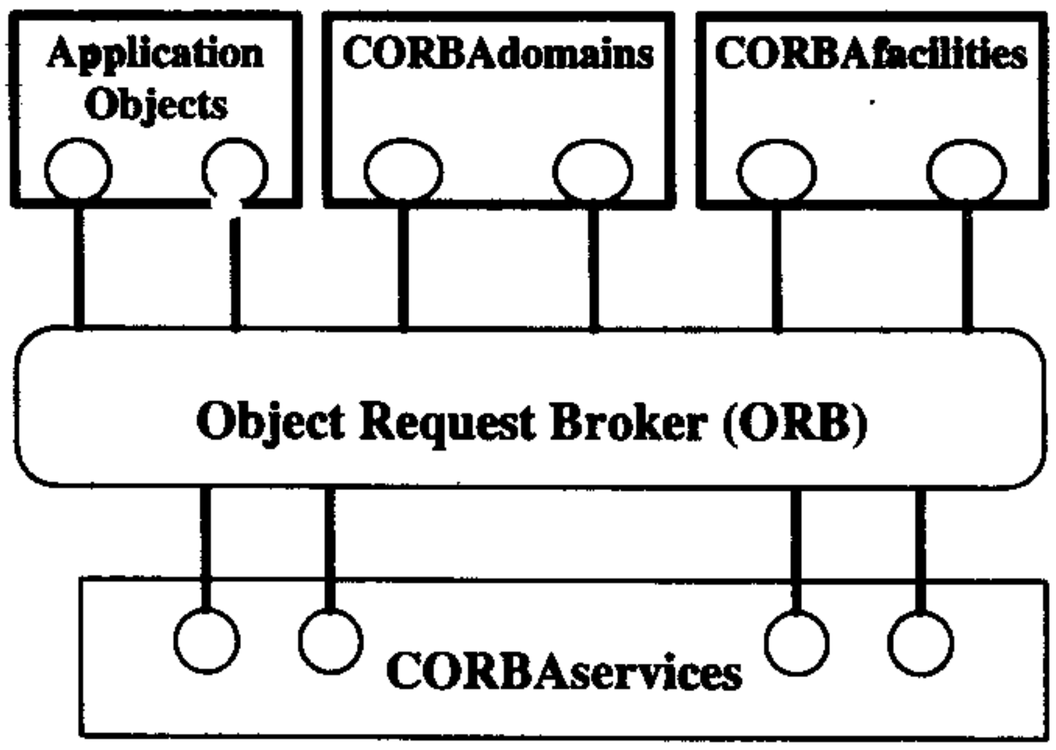


Figure 2.1 Object Management Architecture

CORBAservices are a set of enabling interfaces for globally applicable service capabilities. Some of the services are Naming, Persistence, Security, Trader, Concurrency. CORBAdomains include CORBAfinancials, CORBAmed, Internet, Business Objects, Manufacturing. CORBAfacilites capture any of the domain needs that are in common across multiple domains. The Applications can use standard interfaces from other parts of the architecture. The Applications(clients) can invoke methods present in Objects(servers) present anywhere on the network through ORB. The ORB makes the network/language used for implementation/operating system

transparent. Mobile Agents is one of the facilities that is going to be added to CORBAfacilities, work is going on for standardising the Mobile Agent facilities. An Application developer can concentrate on the functionalities to be provided by the application and then inherit the Transaction, Security and other services provided by the ORB, thus producing a made-to-order application.

2.2 Telescript Technology

Telescript was the first system that brought mobile agent technology into commercial market. This system is General Magic's product. Telescript is the language used to create agents to be deployed in Telescript agent systems. Telescript is an object-oriented language, which treats everything as objects. An object's interface consists of attributes and operations, access controls like public and private allow an object's interfaces to be accessible or not, to other objects. Parameters to operations are objects and operations can throw exception objects, which is to be handled appropriately by the object invoking the operation. An object's implementations consists of properties and methods. Properties represent the internal characteristic of an object. Methods implements the operations. Objects are manipulated by references. References can be of two types, protected and unprotected. A method can manipulate an object, to which it has unprotected reference.

Telescript technology models a network of computers as a collection of places. A place is the context, where service is provided. Agents and Places represent some Authority. Authority is needed to control access to various resources. Authorities limit what an agent or place can do by assigning permits to them. Permit is an object which grant capabilities. An agent or place can never increase its capabilities, and the capabilities are inherited by the agents they create.

The agent hosting system consists of Telescript Engine, which implements the Telescript language, the engine is the VM for Telescript. The VM executes the language instructions by making API calls into the operating system on which it operates. The VM uses the resources of the host for storing agents in persistent storage, transporting the agents and to enable the agents to interact with applications written in other language like C.

The language's important feature is that, it provides a single statement **go** and the place name as the parameter. Whatever is the communication infrastructure followed, is not visible outside, all the necessary API calls are made by the VM. If an exception occurs, because of lack of network connectivity or for any other reason, corresponding exception object is thrown.

The system also provides **meeting** for agents to meet, with data which specify the agent to meet and also some other constraints(the other agent must be in the same system). If the meeting can not be arranged an exception is thrown, which the agent handles.

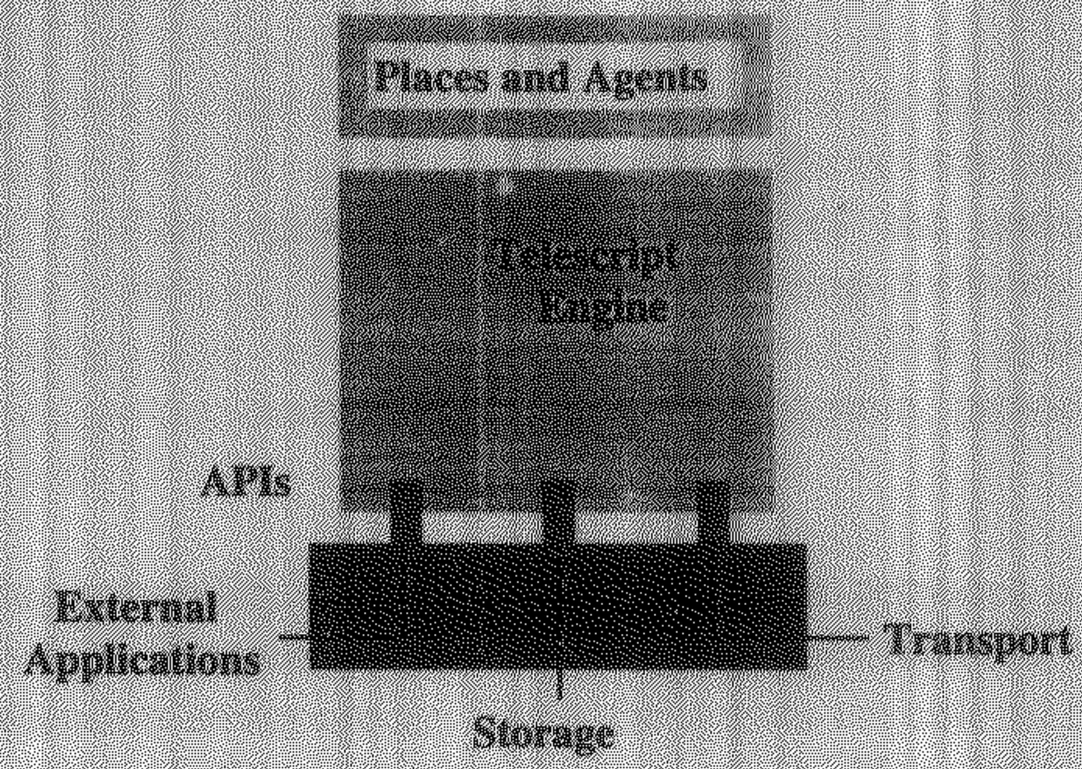


Figure 2.1 Telescript Architecture

Even though being the first commercial product in this field and the implementation made it very easy to use, because of wide spread acceptance of Java, General Magic withdrew this product from the market. They introduced Odyssey, an implementation in Java which provides many of the features of Telescript, like places, meetings. Odyssey also provides support for IIOP, DCOM, RMI. This is the only system which provides support for many agent transfer facilities. Odyssey is a pure Java implementation, it requires JDK 1.1

2.3 ARA

ARA(Agents for Remote Action) system's main idea is to develop a platform for supporting mobile agents to move and execute without any interference, using various languages, existing applications independent of the underlying operating system and hardware details. The system tries to bring mobility to the existing programs and programming environments(like C/C++). The system acts as the middleware between the applications and the hosting system, providing facilities for mobility and execution.

In order to provide portability and security, the system provides a virtual machine equipped with a set of interpreters for various languages(that are supported by the system) and an execution environment. Since the agents execute in this environment, the underlying details of the operating system are hidden from the agent, thus providing portability. The agent's activities are restricted within the execution environment.

Mobile agents are programmed in specific interpreted language and executed in an interpreter for this language, in a special execution environment called Core. The Ara system embeds the language specific features in the interpreter and the language independent functionality in the Core. The language specific agent state information storing/restoring is built in the interpreter and general state information independent of the languages needed for moving the agent are embodied in the Core. Hence several interpreters for specific languages can be used on top of the Core.

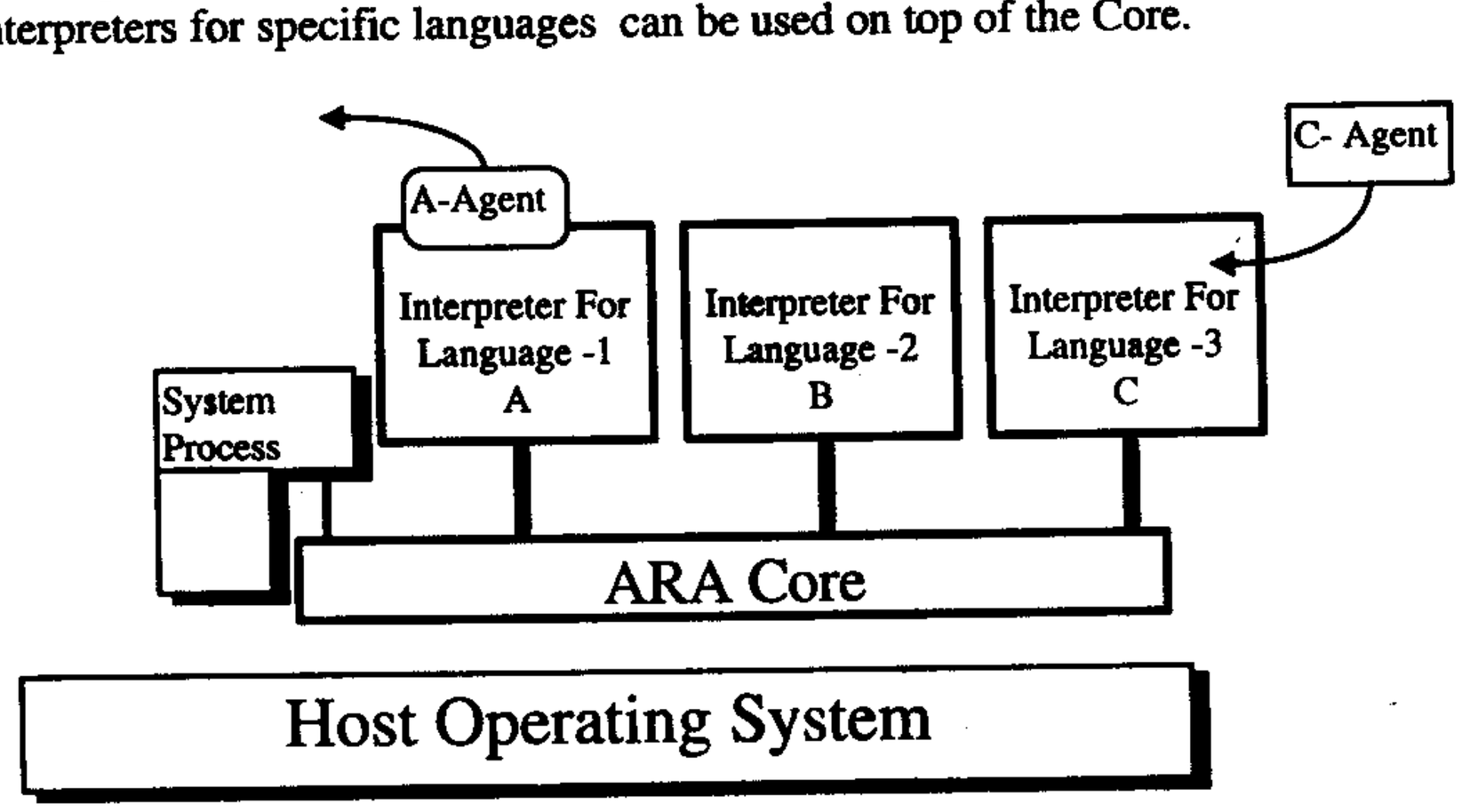


Figure 2.2 Ara Architecture(Taken from [1])

Ara has interpreters for Tcl, C/C++ in the Core. Work for adding Java in the Core is on. Since C/C++ is not interpreted, in order to support the widely used language and existing software, the Ara system designers designed a special interpreter for C/C++ specially for mobile code support. This interpreter first compiles the source into portable MACE (VM for C/C++) bytecodes, which is interpreted by MACE interpreters.

Each agent in Ara system is a process, with its own state and progress or behaviour. Creation and deletion of new agents and transferring them to some other systems are some of the basic functions offered by the Core. Every agent is assigned an unique immutable name on creation. Each agent after being created will execute on their own, like other active agents. Agents can clone, suspend, resume, destroy themselves, sleep. Since the agent moves from one system to another for achieving its goal, some of the service providers may charge for the resources used by the agent, hence the creator of the agent sets some limits on the resources the agent can request for. The agent system also restricts the amount of resource given to an agent depending on the authority of the agent. The agent may return the allowance back thus entitling for later request grants. The system process, which are compiled privileged process, can directly access the facilities provided by the Host Operating System. These are present for internal purposes, and also helps in modularising the system. Various higher-level services, like searching a Database is provided by compiled Agents termed **Server Agents.**

Ara system provides various forms of agent communication, the agent may contact another agent present on a remote system or else move to that system and communicate locally. Ara system encourages the local interaction. It provides various schemes, shared memory, direct message exchange, special procedure calls, disk files.

One of the major feature of the Ara system is that it provides restoration services. An agent can ask for a checkpoint - record of its current execution state, any time during its execution. The checkpoints are stored in persistent storage. The agents are programmed, so that when they are trying to do some risky operations during which its state may be lost, it creates a checkpoint and then tries the operation, if the operation fails, from the checkpoint the agent's state can be restored.

All agent transfer request, whether inbound or outbound are handled by Communication process. Two Communication processes present at different system communicates and passes the byte stream, that represent the state of the agent being transferred. The receiving Communication process sends this package to the core to restore the agents state. At present TCP is being used for transport, SMTP support is planned. Thus the communication process can choose one of the available alternatives for agent transport.

2.4 Agent Tcl

Agent Tcl, uses the Tcl scripting language. Since Tcl is an easy to learn scripting language which presents its operations at a high-level, users are provided with easier methods for deploying agents. Agent Tcl agents are extensions to Tcl. The main idea of Tcl is to allow programmers to group various applications and utilities to provide a solution. Agent Tcl agents are also provided with the standard Tcl commands at the system. The extension provides the agents the facility to migrate from one machine to another, create other agents, to communicate with other agents, to obtain knowledge about the environment in which it is executing. Since Tcl is a glue language, various applications can be embedded with agent specific commands, thus producing agent enabled applications.

Agent Tcl systems main goals were: 1) To reduce the transfer to a single instruction, and allow this instruction to occur at any point of the application. The system should implement in such a way that the state of the agent is captured and sent to the target machine, whenever needed. All this should be done transparently to the user(or programmer). 2) To provide communication mechanisms that are flexible and low-level, but that hides the transmission details. 3) To provide a high-level scripting language as the main agent language, also support multiple languages and transport mechanisms, and provide facilities for future addition of languages and transport mechanisms. 4) To provide effective security.

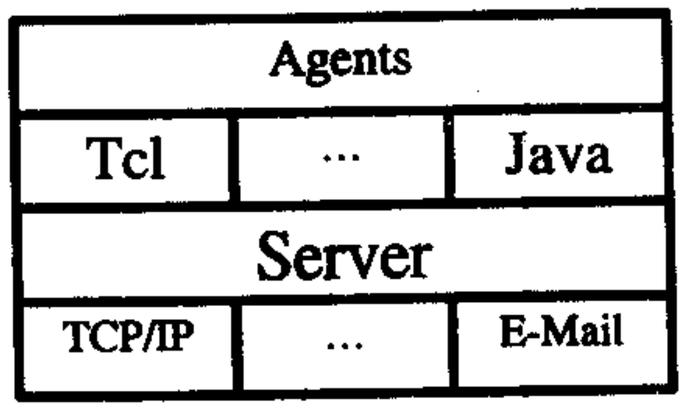


Figure 2.3 Agent Tcl Architecture(Taken from [1])

The above figure shows the architecture of the Agent Tcl system. The architecture has four levels. The lowest level provides API for the available transport mechanisms. The next higher level is the server that runs at each network site which provides agent services. The next higher level consists of interpreter for each language supported. At the higher level is the level at which agents execute.

The server keeps track of the agents running on its machine and accepts incoming agents after proper authentication, passes the agent to the proper interpreter. The server also takes care of transferring an agent to another system. The server also maintains a name space, so that each agent is identified by unique names or Ids. The agents can communicate with each other by the server's messaging service. The server provides ***event*** messaging and ***connection*** messaging. The server also provides the agents the facility to back up themselves.

Other services like access control, resource discovery, group communication are provided by agents. Two special services provided are the ***docking*** and ***resource management*** service, both managed by agents. The docking service supports disconnected operation, if due to some error an agent is not able to migrate to its target machine, it is added to queue maintained by dock agent , which moves the agent once transfer is possible. Resource manager agents along with encryption system and language-specific security models(Safe-Tcl) guard access to critical system resources. The access restriction is based on authentication.

The interpreters used contains the interpreter(core Tcl) itself, the security module which prevents agents from malicious behaviour, state capturing and restoring module, and a set of API to interact with the server to handle migration, communication and checkpointing. The security module of Tcl agents is the Safe-Tcl extension which allows a Tcl interpreter to replace potentially dangerous commands with safe equivalents that performs access checks.

Another advantage in using Tcl as the agent language is that, Tcl is combined with Tk, the tool kit for creating professional quality user interfaces. Tk provides a simple scripting language to create high quality interfaces, with less effort. Agent Tcl system along with standard Tcl will also provide the Tk commands, hence an agent can carry Tk commands to interact with an user at the target machine.

Agent Tcl is a result of ongoing research project in Dartmouth College and is under continuous development. Agent Tcl like Ara System uses Tcl as the main agent language. Compared to other languages like Java and Telescript which are Object-Oriented, Tcl allows rapid development of small to medium-sized applications. Tcl is also slower compared to other scripting languages, interpreted bytecodes and native machine code. Since only code modularization possible in Tcl is procedure, as the application becomes bigger, it is difficult to maintain. Agent Tcl does not yet support multiple languages.

Sun Microsystems who ported Tcl to various platforms, has introduced extensions to Tcl called Jacl , using which Java code can be controlled by Tcl scripts. Thus applets embedded in web pages can be controlled by Tcl scripts if the browser supports Tcl engine. Thus Java and Tcl extension for Java(Jacl) can bring the best in both . It remains to be seen how this extension may be applicable in Agent Tcl system.

2.5 Aglets

Aglets is a framework written in Java to facilitate development of mobile agents. The name aglets is a play of words Agent and Applet. The Aglet model mirrors a lot of models in Java, Applet, beans, AWT call-back. The Aglets model reflect the Applet model in many ways, like an Applet having init, start, stop, destroy, the Aglet has clone, destroy, despatch. The Aglet also has AgletContext similar to AppletContext. The Aglet model also has methods like getAudioClip, getImage.

The framework contains a base mobile agent class called aglet. The system provides a globally unique naming scheme for agents, it uses itinerary for specifying travel patterns with multiple destinations and automatic failure handling, a white board mechanism allowing agents to collaborate and share information asynchronously, message-passing scheme that supports asynchronous and synchronous communication between agents, network agent class loader that allows an agent's byte code and state information to travel across the network and an execution environment. Agents can invoke methods in other Agents only through the AgletProxy objects. When invoked the AgletProxy first checks with installed SecurityManager whether such an operation can be allowed.

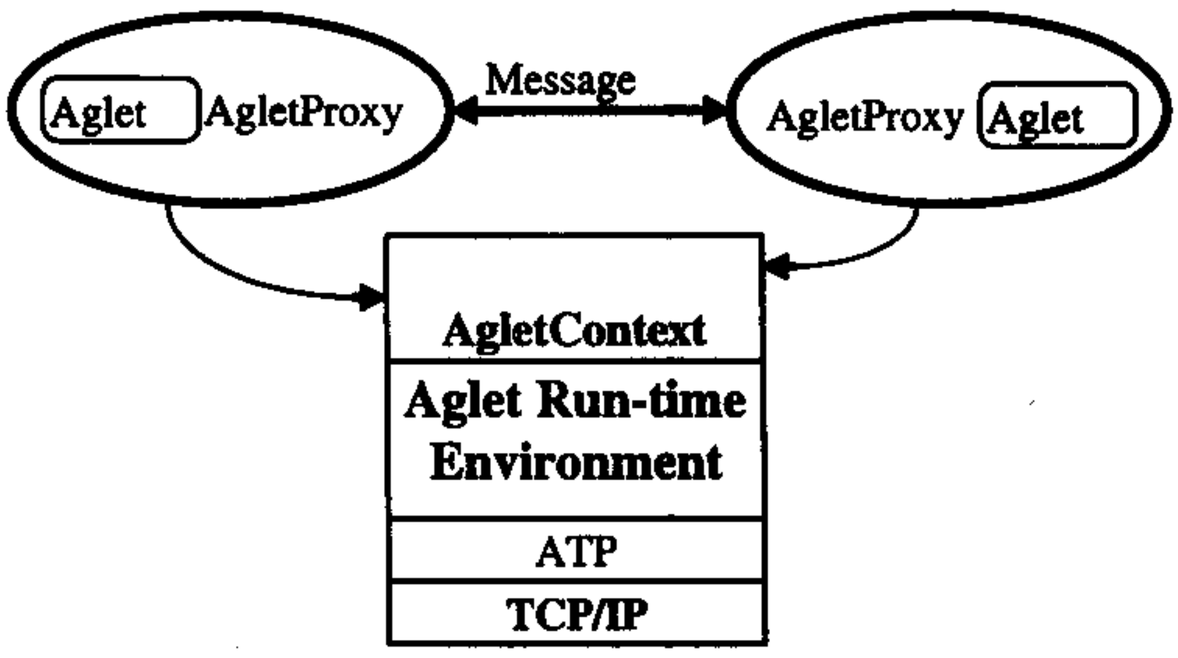


Figure 2.4 Aglet Architecture

The framework is part of Aglet Work Bench, which also includes Tahiti, a Visual agent manager, for keeping track of various agents executing in the system, a Visual agent builder to increase the productivity. Using a drag-and-drop mechanism, the builder Tazza makes it easier to create agent applications with mobile behaviour and also visual front ends. If the system is deployed in Corporate offices, then access to databases will be needed. AWB offers special packages for data access, including JDBC/DB2.

For transporting the agents across the systems, the framework provides ATP (Agent Transfer Protocol). ATP is an application-level protocol for distributed agent-based information systems. ATP offers a uniform and platform-independent protocol for transferring agents between networked computers. Even though mobile agents may be written in many different languages and for a variety of agent systems, using ATP offers the opportunity to handle agent mobility in a general and uniform way. ATP also

provides a uniform agent transport mechanism and allow a standard agent query facility to be used. This ATP package is implemented in Java, and provides API for creating ATP daemons, connecting to ATP sites and generating ATP responses and request.

AWB also includes Fiji, a Java applet based on aglets framework, capable of creating an aglet or retracting an existing aglet into the client browser. Like other applets, the required classes will be dynamically downloaded to the browser as it is needed. Web sites can also provide ATP daemons, along with Web servers, to support aglets, then Fiji like applet can dispatch aglets to the Web site for remote search or disconnected execution at the Web site.

Security in Aglets framework contains three layer. The first layer is provided by the Java language itself, the Java runtime environment's bytecode verifier performs a series of checks to ensure that the code is in correct format. In the next layer is the security manager, which provides the users the facility to implement their own security policy. The third layer is provided by the Java's security API, which provides cryptography, digital signature which makes it easy to include the security functionality in their agents. The framework is also compatible with JDK-1.0.2, with the RMI support added.

2.6 Mole

Mole Agent system uses Java to implement the system and also to develop agents. In this system agents are modelled as clusters of objects without reference to the outside, except to its execution environment. The agent system consists of a set of ***locations***, which are places where agents are executed. In locations, agents can compute, communicate to other agents and also can use resources and services offered by the system. In Mole system, the resources and services are represented by ***system agents***, which can used by other agents called ***user agents*** by communicating with system agents. Each agent location has a unique name, DNS names are used as location names.

Mobility can be implemented in two different ways, remote execution and migration. In the first one a new agent is started at a specified location, whereas in migration, the agent continues its execution at a different location. In Moles system, the system agents are stationary agents and the user agents migrate between locations.

The migration provided by the system is weak, that is only data state and program state are transported to the destination location. When agent executes the migrate statement, a stop method is called which agent implements to have a say in deciding what to transfer. Communication between agents are provided by RMI and messages passing between the agents.

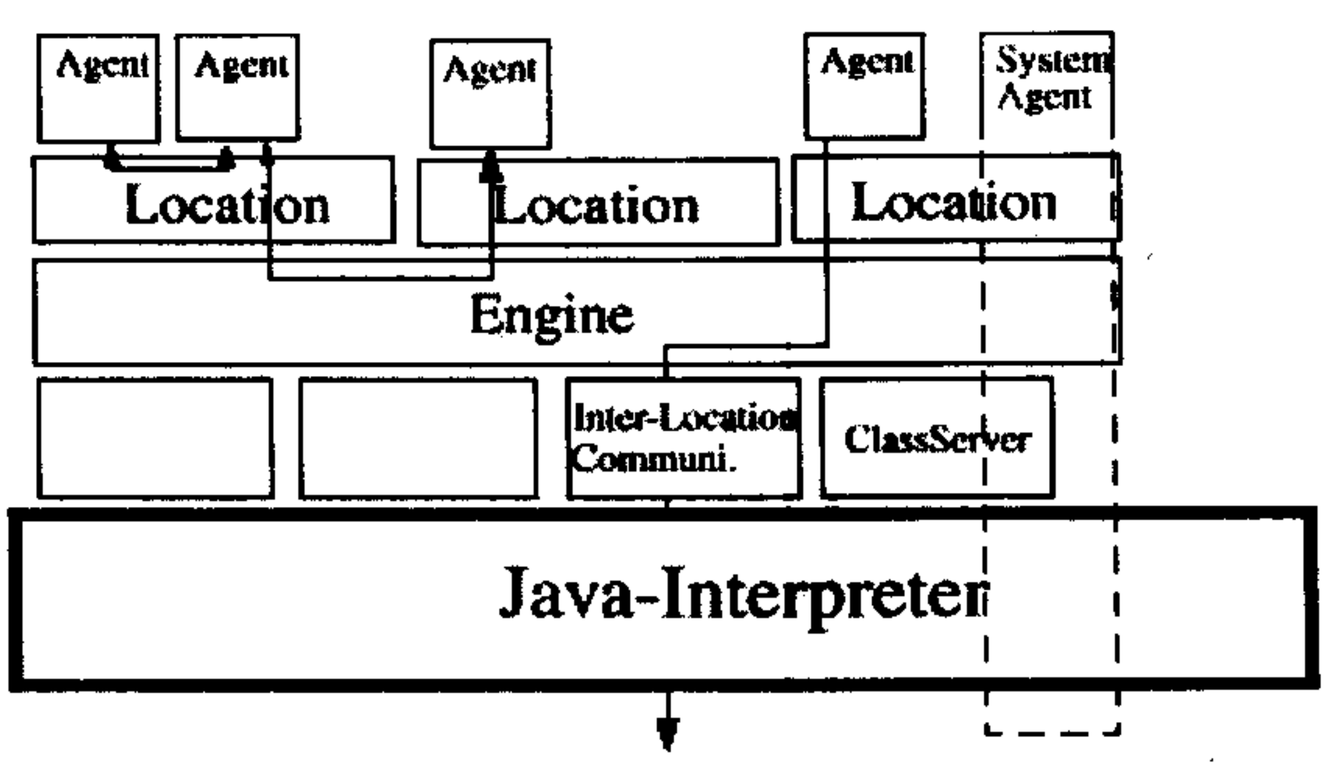


Figure 2.5 Mole system Architecture

Agents are executed on a location, using services provided. The user agents communicates with the system agents to get access to resources and services. System agents may also offer access to legacy applications, using interface to C/C++ offered by Java. The Location manages the agents executed at the location. It starts the agents on its arrival and provides services like migration, communication, directory and other specific services. The Engine manages the Locations executed on one system. It offers the Class server and inter-location communication. The class server is responsible for getting unknown classes needed for agents about to be executed or agents executing on a location of the engine.

KOSAgent Architecture

3.1 KOSAgent : JAF

KOSAgent is a Java based Agent Framework. The Framework facilitates Hosts to provide mobility to processes(or Agents). The Framework uses the security mechanisms provided by Java, for Authentication and to prevent alien code from misusing the system. The Framework provides Agent Transfer mechanisms, through use of **KATP** (KOS Agent Transfer Protocol)(**implemented in this release**) and/or as CORBA service(**not implemented in this release**). The Framework relies on CORBA ORB for inter-Agent System communications.

The Framework provides centralised administration of various AgentSystems(possibly distributed over a network of various machines) providing agent services. If agent services are decided to be provided by a network, various AgentSystems can be deployed across various machines in the network, but the internal structure of network is not made visible outside, all that client sees is a Host, termed AgentHost(explained later) which represent the network(or its Authority) and handles the service requests through the GateManager. Various systems(AgentSystems) providing the services can be started individually, once each system starts, it informs the AgentHostManager (through service provided on top of CORBA ORB), that it is ready for accepting service requests. Thus AgentHost can provide information to clients about various services available, and accept service requests, and place the agent in appropriate system. All AgentSystems communicate with each other or with the AgentHost through CORBA ORB.

Since Agents cannot carry around all the Code that it may need in its life time, ObjectManager of an AgentSystem serves classes(custom code which will not be present in a target AgentSystem) to other AgentSystems, where its agents are visiting, on demand. Classes needed by an Agent are packed and served as Jars(Java Archive-ZIP files), which are to be signed by the Authority to whom the Agent belongs. KCTP (KOS Code/Class Transfer Protocol) is used by AgentSystems for serving or requesting classes. Whenever an Agent enters an AgentSystem, it is equipped with a list which provides the locations of the code that it may need. Various policies can be used to obtain the classes, either all the classes can be obtained at a single go, or classes are obtained whenever the Agent requests during its stay in the system. The Policy to be followed depends on the Administrator of the AgentSystem.

Code needed by an Agent can also be served by Web servers. This method of class serving helps the mobile users, who after sending their Agent may have to disconnect from the Internet. The framework along with providing functionality to obtain classes from ObjectManager of an AgentSystem(by KCTP), also provides an option to obtain code from Web Servers trusted by both Service Provider and Client. The policy followed is again based on the decision of Administrator.

The Agent service providers can provide custom services, by extending the base services provided. The Framework does not handle charging for services, though it does impose certain restrictions on the amount of resources an Agent can enjoy, by the existing policies designed by the Administrator.

The Framework also provides a Web interface for clients to request agent services from Agent Hosts. So a Agent Host System Administrator can provide Agents to enter the system from outside and/or Web interface for clients to request for agent services and send agents to client side(browser) with the results through Web interface.

The Framework, does not provide any kind of Agent Communication Language. The Clients after retracting their agents, can view the results of their agents history and other results through GUI, provided by the Framework. Each AgentSystem can act as an independent Service Provider or after start up can register itself through ORB with the AgentHostManager. Thus in an AgentHost System, with various AgentSystems distributed across the network, the GateManager at the AgentHost machine, receives the requests and checks with AgentHostManager whether the client can be granted the entry permission, and if permitted accepts the Agent and passes it through the ORB to the GateManager present at AgentSystems.

3.2 Terminology

Place:

Places are the execution context of Agents. The Agents interact with the AgentSystem or other Agents through the place in which they are executing.

AgentSystem(AS):

AgentSystem consists of one or more places. The AgentSystem provides various services, like Directory, Query, Externalization, Transport, to agents. Places help an AgentSystem to balance the load. In a system with multiple processors, each place may be executing on each processor. Each AgentSystem represents some Authority. AgentSystems are characterised by their type, which indicates what language it supports for Agent development. In this Framework AgentSystem implemented use Java as Agent Development Language. Other AgentSystems of different type can use the internal communication services provided on top of ORB to interoperate. Thus another type of AgentSystem can coexist in the AgentHost system, thus having the GateManager(on AgentHost machine) of this Framework to handle its requests.

AgentSystemManager(ASM):

AgentSystemManager, creates more places when necessary. It keeps track of various places and maintains an Agent System. It also provides a UI for interacting with Agent System Administrator. The Administrator or User of an AgentSystem interacts with his AgentSystem , keeps track of the status of AgentSystem, create/retract/dispose/send Agents, view the returned Agents through UI provided by AgentSystemManager.

Region:

Various AgentSystems may be of different types, but of same Authority are grouped together and termed as Region. Again Region provides the facility of load balancing. If more request for AgentSystem service arrives and if all the AgentSystems are already heavily loaded or if a particular AgentSystem is overloaded, then the Region Administrator can spawn another Agent System to balance the load.

RegionManager(RM):

RM is to AgentSystems as ASM is to Places. Region may be spread across various machines in the internal network. This structure is transparent to the Agents.

AgentHost(AH):

AgentHost is the machine which hosts various AgentSystems. In a way, Agent Host acts as a Firewall to the internal network consisting of various Agent Systems. The client who requests for Agent services, knows only about the Agent Host and various Agent Systems provided by that Host, and not the internal distribution of the

AgentSystem(across machines). AgentSystem can also individually provide services directly by spawning its own GateManager.

AgentHostManager(AHM):

AgentHostManager, starts the AgentSystem, when the Machine starts. AgentHostManager controls the entry of agents from client machines, based on AgentHost level policy and Region/AgentSystem level policy. Even if AgentHost Level policy forbids entry to agents from particular Domain, if the AgentSystem level policy does not enforce such restriction, then the AgentHostManager permits entry.

ObjectManager:

Since an Agent cannot carry all the classes, it may need, the Client's System (AgentHost or AgentSystem)provides ObjectManager which serves all classes needed by an Agent in its Travel.

Services:

Services provide the Agents access to various resources. Some of the Services are Directory, Messaging, Query, Immigration etc. In this Framework Services are implemented as Agents. Thus Directory Service is provided by Directory Service Agent.

Directory Service:

Directory Service gives information about various services(agents) available on the system and also information about various agents registered for providing various services.

Query Service:

Query Service gives information about various services available or about status of an agent.

Immigration Service:

Immigration Service provides the mobility to the agents. The agents move from one AgentSystem to another AgentSystem through Immigration Service.

Messaging Service:

Messaging Service provides ways for agents to communicate between them.

Special Service:

An AgentSystem may provide other services like Booking, Selling/Buying of

goods etc.

Events:

In an Agent's lifetime various events happen, this Framework uses a Event Delegation similar to the one provided in Java.

JAR:

Jar stands for *Java Archive*, which is nothing but ZIP format files. Classes are packed in Jar files. Thus instead of downloading each class separately, all the necessary classes are downloaded in one go, thus reducing the overhead involved in connection for each class.

KATP:

KOS Agent Transfer Protocol, is an Application-level protocol used for transfer of Agents from one AgentSystem to another. The protocol is used by AgentSystems to transfer(or provide mobility) an Agent from one System to another and query about the status of an Agent and request for some services.(More in detail in Implementation Details Section).

KCTP:

KOS Code Transfer Protocol, is an Application-level protocol used for requesting /serving resources(Jar files containing Agent support classes) . (More in detail in Implementation Details Section).

3.3 Architecture

The Architecture of KOSAgent Framework is shown in the diagram below.

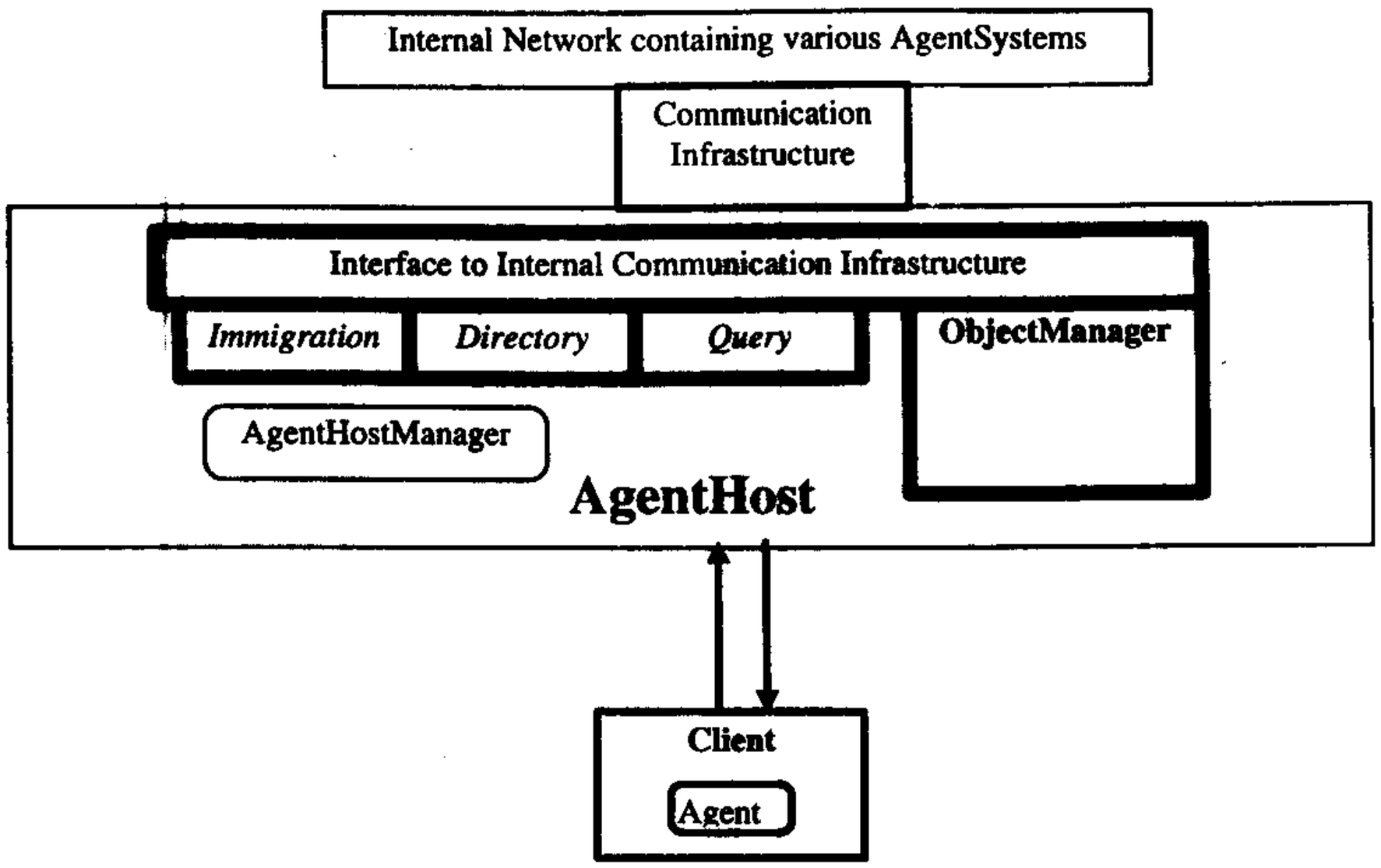


Figure 3.1 AgentHost as seen from the Client side

The above figure shows the view of an AgentHost as seen by the client. The AgentHost provides the basic services as illustrated. It also contains the ObjectManager, which serves code their Agents visiting another AgentSystem may need. The AgentHost provides a TCP based KATP for Agent Transfer. The AgentHost communicates with the Various AgentSystems present in the internal network through the Communication Infrastructure(based on CORBA). On the Client side, the AgentManager which creates Agents on the Clients request is responsible for despatching and retracting the Agents.

The next diagram shows AgentSystems consisting of various number of Places with many Agents. Many AgentSystems may be present in the same machine(physical). The AgentSystems communicate with the AgentHost through the Communication Infrastructure. Each AgentSystem runs on separate Java Virtual Machines. Some services provided by the AgentSystems may be represented by a type of Agents called ServiceAgents(**not implemented in this release**). These ServiceAgents also execute in the Places, but they have more privileges. The AgentSystems uses the Communication Infrastructure for transferring Agents out of the Host or for getting information about other AgentSystems present in the Host or to load some class from the ObjectBase. Various AgentSystems belonging to the same Authority are logically grouped as Region. AgentSystems belonging to a Region can be present in different machines(physical).

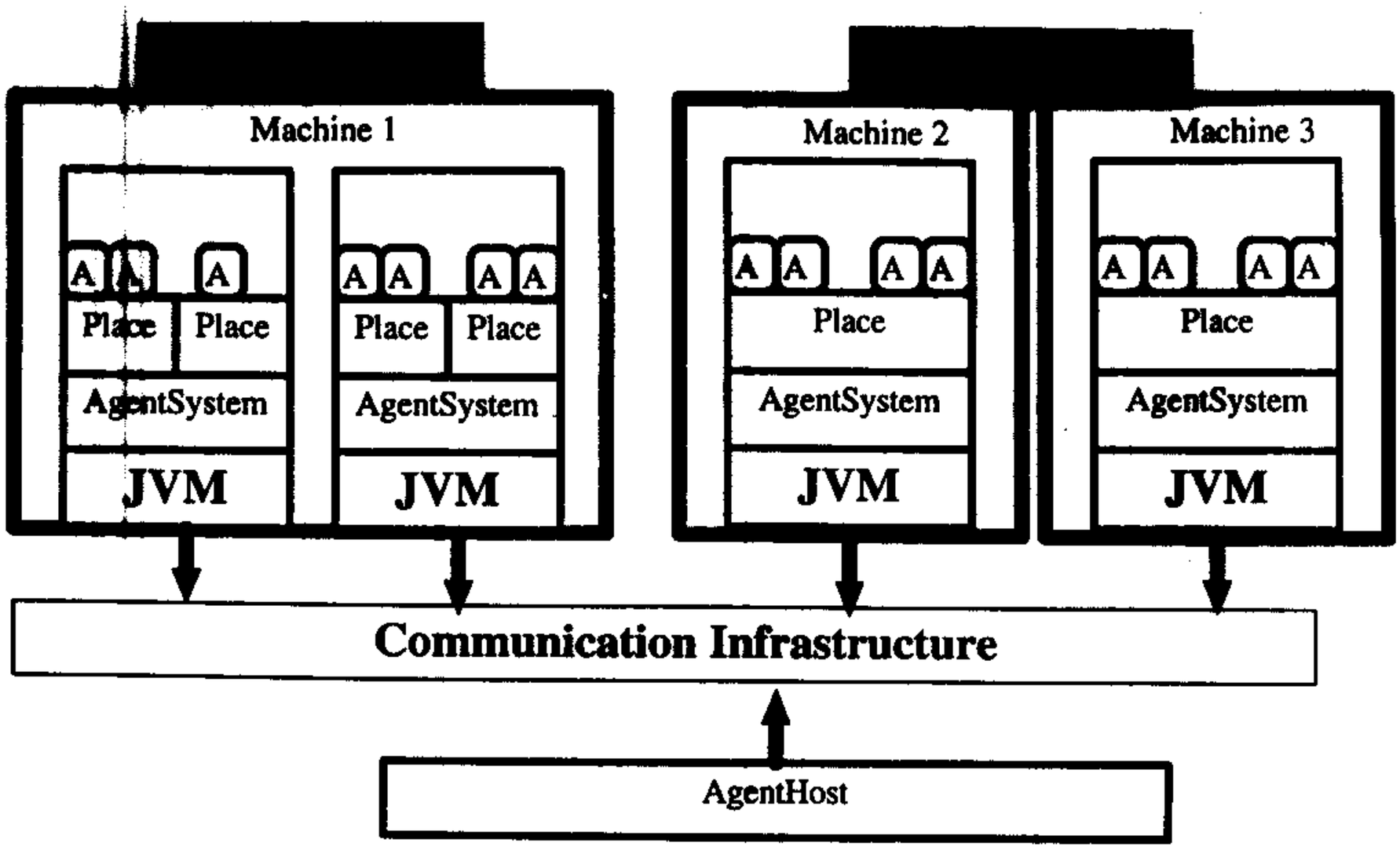


Figure 3.2 AgentHost-AgentSystem Communication

The AgentSystem provides various services to the Agents executing in various Places present in the AgentSystem. The Agent requests the AgentSystem for various services like Immigration, Querying, Directory, Messaging and other Special Services through the Place in which it is executing. The diagram below illustrates this.

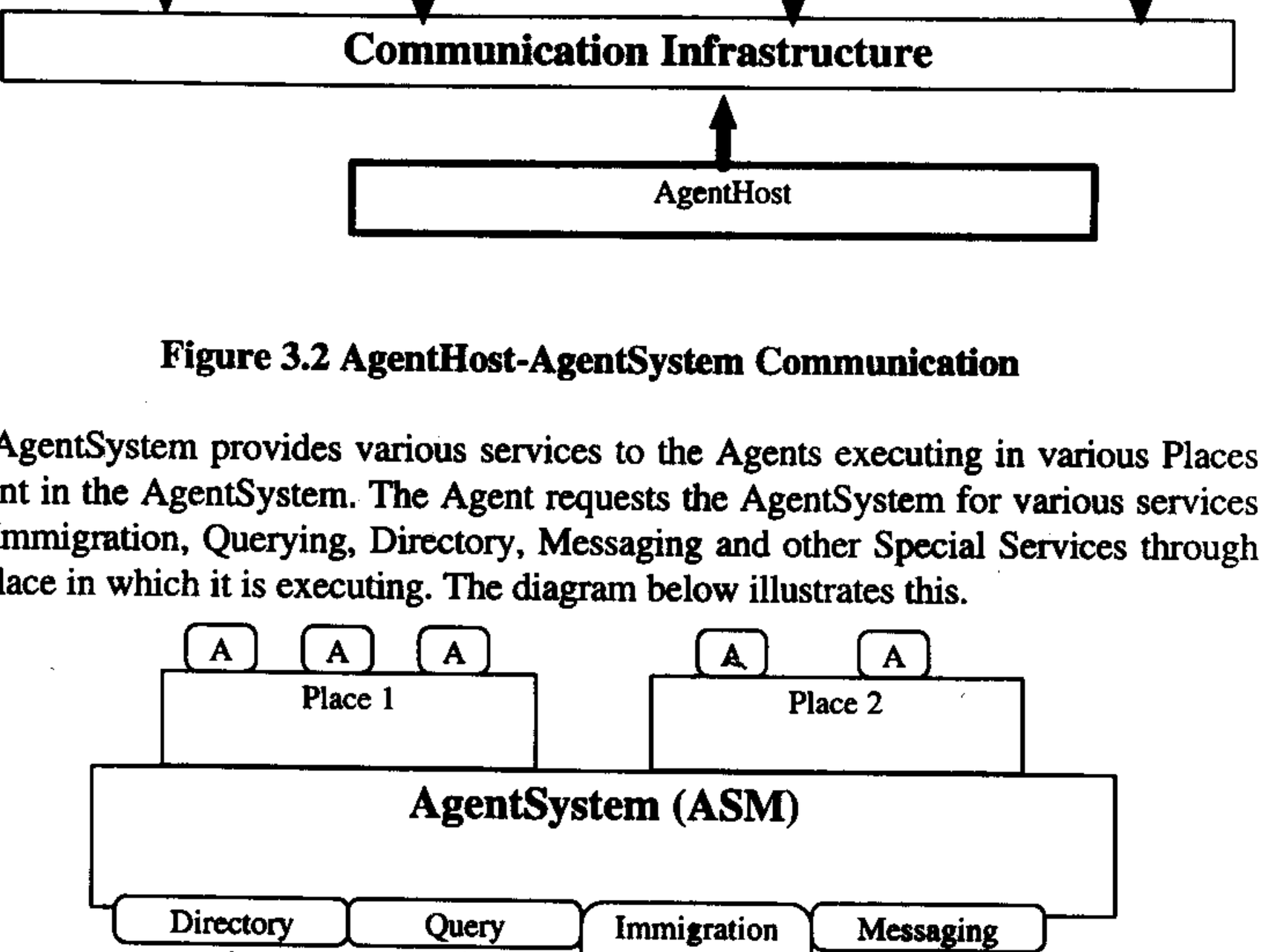


Figure 3.4 AgentSystem Architecture

The ObjectManager maintains code for serving in Jar format. The Jars will be digitally signed by the Authority to whom an Agent belongs. Since Services provided at an AgentSystem for various purposes like searching Database of information, the Agent has to carry minimal code to learn what services are available, and how to invoke the services. The Agent has to carry only the result back. Since the functionality needed for most purposes will be provided as Service, the Agent has to carry around little code. Custom extensions that the Agent needs to have to complement the service, will be very little.

3.4 Features

- Provides a Firewall arrangement(AgentHost), through which internal structure of the network can be hidden.
- Provides Agent Transfer over sockets(Katp) and/or as a Service over CORBA ORB.
- Provides ObjectManager for serving of classes(by used of Kctp) needed by an Agent at its destination AgentSystem.
- Classes needed by an Agent can also be provided by Web Servers. Since users who connect to the Internet through Dial-Up services may have their Web pages hosted on some Web Server, the classes can be made available there. The users can disconnect (no ObjectManager will be running) after sending their Agents, even then classes are provided by the Web Server, ensuring that the Agents will not be without the required resources.
- Since the Framework is built with Java and CORBA is used for internal(AgentSystem-AgentSystem and AgentSystem-AgentHost) communication, platform independence and network/language/hardware transparency is assured.
- Other kind of AgentSystems, with appropriate bridges can use the GateManager, for providing services to clients, provided by AgentHost, through CORBA ORB.
- Provides Centralised control, through AgentHostManager and RegionManagers thus allowing for Load balancing.
- Provides the Administrator the option whether to allow Agents to migrate in or just create Agents based on Request.
- Provides security through use of Digital Keys and Certificates for Authentication of Servers and Clients.

Implementation Details

4.1 System Implemented

The present implementation provides AgentSystems which individually provide services to incoming agents through GateManager. An AgentSystem when started reads various properties, specified later, from a file and enforces various policies like maximum allowed agents in a Place or maximum number of Places in an AgentSystem. AgentSystemManager started by AgentSystem provides the Administrator or User with UI termed AdministratorControlPanel , through which the User can view the status of AgentSystem or change various properties of the running AgentSystem or create and deploy Agents. Functionalities like AgentHost and various AgentSystems communicating with each other through services provided on top of CORBA ORB is **not provided in this release**.

For transferring the state of an Agent, the Agent instance is serialized using Java's Object Serialization interface(which allows object instances to be written as stream of bytes and also to reinstantiate the object from the byte stream) . The serialized stream which is put into a file is then transferred to target AgentSystem. All the classes needed by an agent is packed in Jar files and served to other AgentSystems.

Mapping of names used for entities used in previous section to explain the Architecture of the KOSAgent Framework are straightforward. Various Important classes used to represent the entities explained before are discussed briefly next.

AgentKOS:

This class is the base abstract class, which Agent Developers are suppose to subclass and provide the necessary functionality.

KOSAgentID:

This class provides Identity to an Agent. ID consist of three parts: Native Host IP Address, User name(or owner), Unique serial number(on Native Host).

KOSAuthority:

This class represents the Owner of an Agent. At present this is not used, later work will have this class encapsulate Owner's Public Key and also few certificates from some trusted Certifying Authority.

KOSAgentSystem:

This class implements the AgentSystem. It also keeps track of Identity of Agents and their status.

KOSPlace:

This class implements the behaviour of Place. This class provides the interface, to Agents executing under their control, for mobility, disposal, messaging and other services.

KOSAgentSystemManager:

This class implements the behaviour of AgentSystemManager. This class is the one that takes care of insertion of incoming Agents into some Place.

ObjectManager:

This class serves out code(classes) to other AgentSystems. At present, it serves code without any authentication of any sort. The code for an agent are put in a Jar file, the ObjectManager serves Jar files, though facilities exist for obtaining specific classes from a Jar file.

GateManager:

This class acts as a firewall to the AgentSystem, it fields all the requests from clients and provides them various services like Entry, Disposal, Retract, Return.

KOSSecurityManager:

This class is a subclass of SecurityManager class provided in Java Core. This prevents Agents from performing actions which are potentially dangerous.

agentClassLoader:

This class is a subclass of ClassLoader provided in Java Core. This class is responsible for downloading classes(custom) that an Agent may need during its stay in an AgentSystem. As mentioned earlier classes can be cached and served through Kctp or Http.

MessageBus:

This class provides Publish/Subscribe Messaging services, where Publishers can only be Administrator.

ImmigrationManager:

This class makes requests to other AgentSystem on behalf of Agents who want to move from one AgentSystem to another.

AgentKOS Structure and its Life Cycle :

The AgentKOS class is an abstract class, hence it can never be instantiated, the Agent developers have to provide proper subclass with required functionality. The class provides a no-argument, do-nothing constructor which is accessible only to subclasses. The subclasses should not override the constructor. Any initialisation that an Agent has to do before it can start executing , is to be provided in a method by name ***onDocking()***, which is called whenever an Agent reaches(or is born at) an AgentSystem.

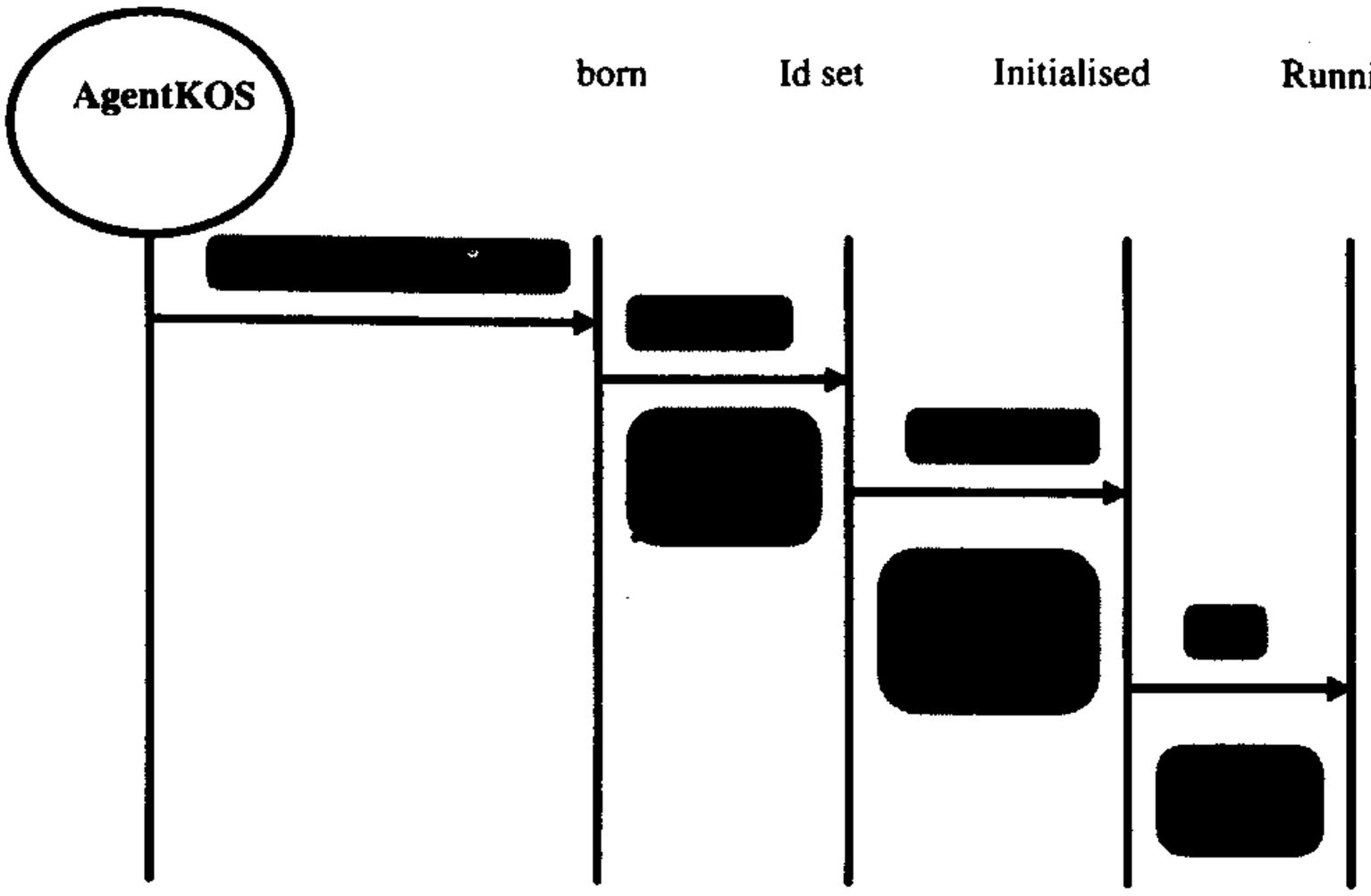
The AgentKOS contains KOSAgentID which gives the identity to an Agent and also a reference to KOSAuthority which specifies the Owner to whom it belongs and a reference to its Home AgentSystem. Whenever an AgentSystem creates an instance(gives birth) of an Agent, after the Agent's Constructor being called, ***onBirth()*** method is called with an identity(instance of KOSAgentID) and its author(instance of KOSAuthority) and home information. The ***OnBirth()*** is called only once in an Agent's lifetime, to provide the Agent its identity and its author and nativity.

As mentioned earlier, Agent Developers are expected to provide code for any specific initialisation in ***onDocking()***, which will be called once instance of an KOSAgent subclass is created and its ***onBirth()*** method has been called. After the Agent being instantiated it is inserted into a Place and a reference to this Place is embedded in the Agent, then ***onDocking()*** method is called after it returns the Agent is allowed to execute by calling its ***run()*** method, in which the Agent Developer provides the behaviour of the Agent.

The AgentKOS class has methods for disposing itself, or retract(return back home) or move to another AgentSystem. The method ***move()***, ***retract()***, ***dispose()***, with appropriate parameters, which provide before mentioned functionality, forwards these calls to the Place in which the Agent is executing.

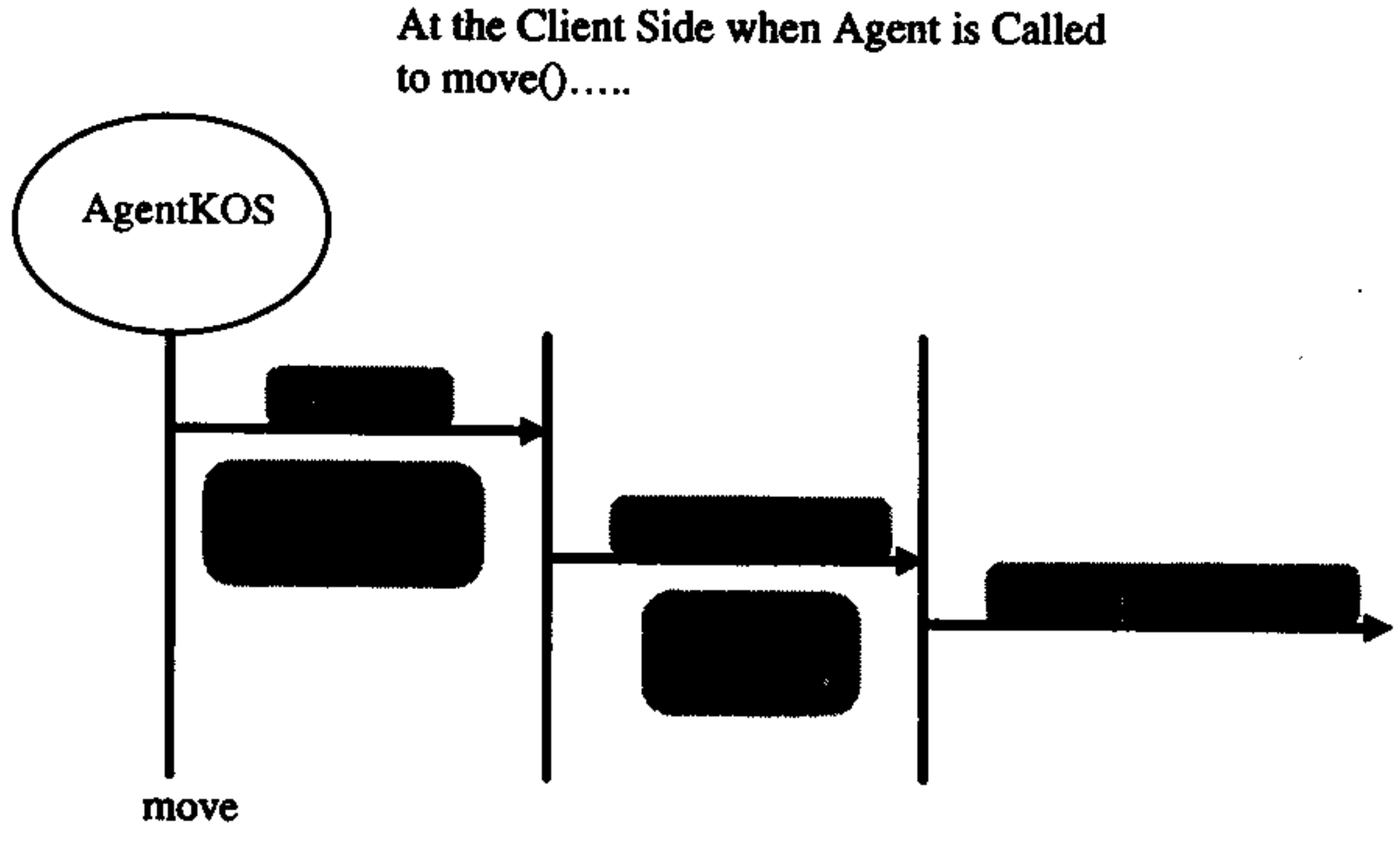
The AgentKOS class has methods ***onMove()***, ***onDisposal()***, ***onRetract()***, which are called before the Agent is moved, disposed, retracted respectively. These methods provide Agent Developers to provide code for releasing resources they may be holding or to prepare the Agent before being moved or disposed or retracted.

Since AgentKOS class implements Serializable interface(provided by Java's Object Serialization API), the instance of AgentKOS can be serialized without much work by the Developer. The reference to Place in which an Agent is executing is declared as *transient* which prevents the Java's Serialization process from serializing it. Agent Developers are better off if they declare any reference to Objects which are not serializable as transient. Even though Java gives the freedom to an Object for writing its own state, it is not desirable, since then the user or developer has to take care of the serialization process.



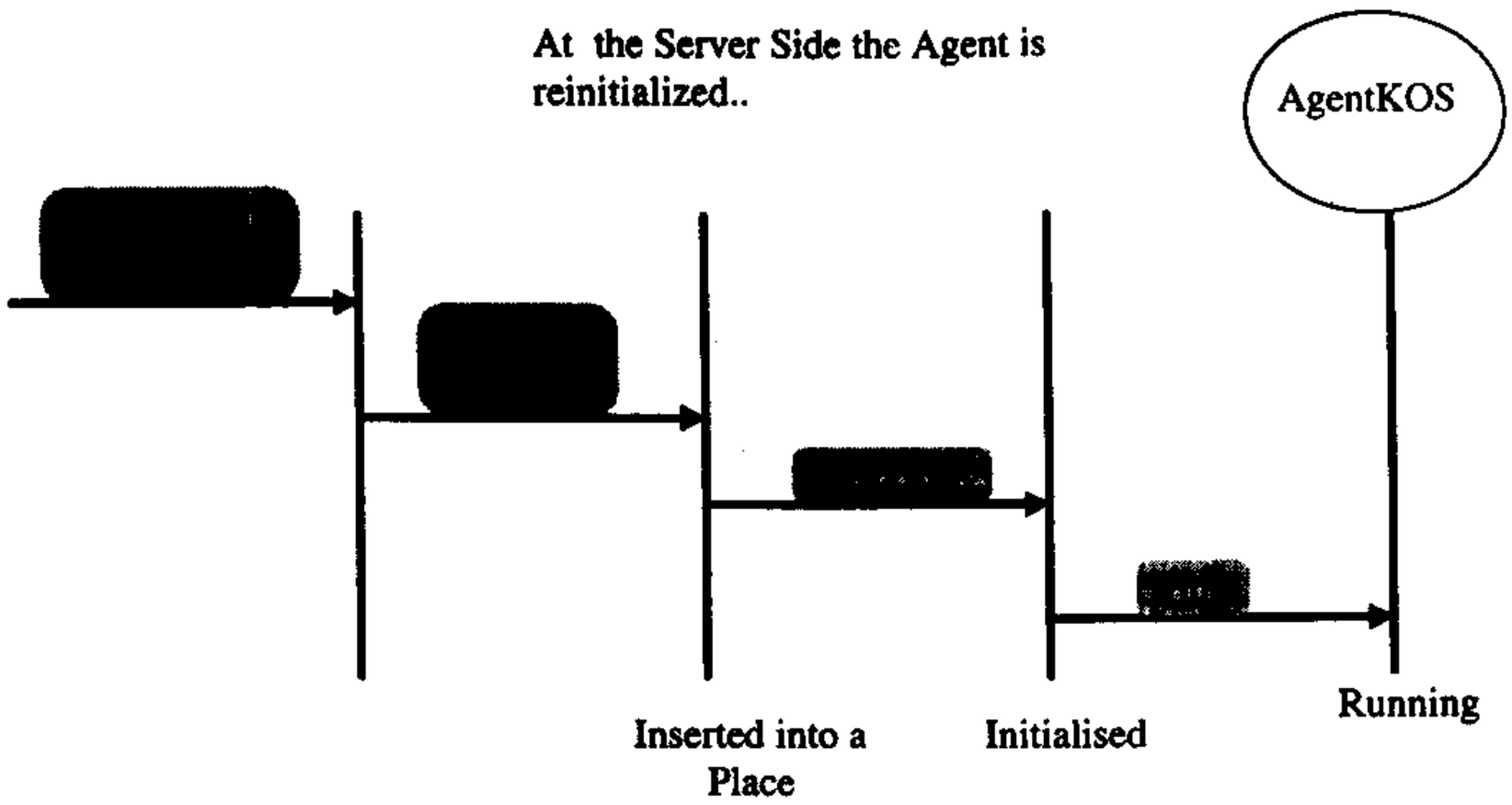
Scenario depicting the AgentKOS initialisation process when it is born.

The next diagram shows the scenario of Agent being packed for transferring it to another AgentSystem.



Scenario depicting AgentKOS being prepared for Mobility on client side

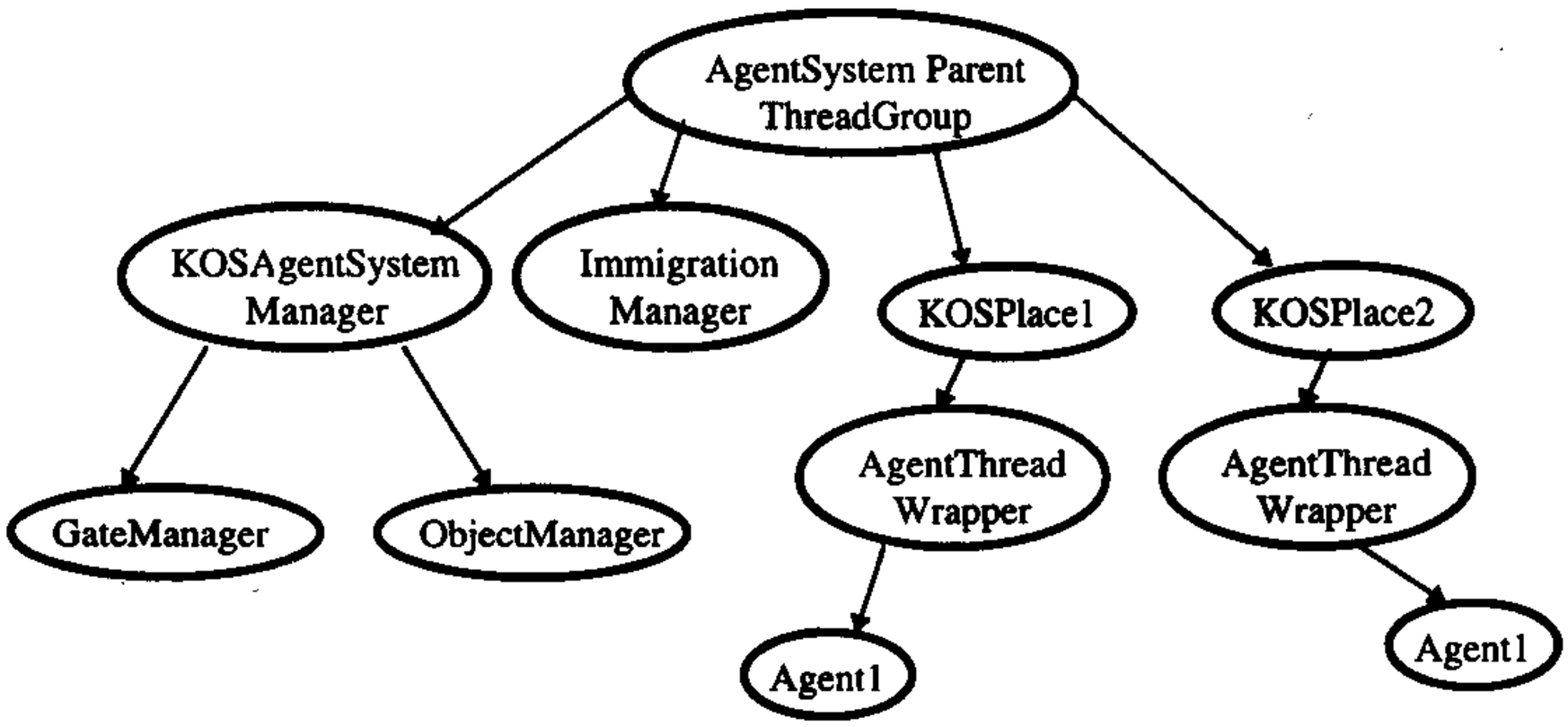
The next diagram shows a scenario where an Agent stream received over the socket is unpacked and initialised and allowed to execute.



Scenario depicting Agent being initialised on the Server side

KOSAgentSystem

KOSAgentSystem class which provides the AgentSystem functionality, during startup reads various properties which will control the execution of the AgentSystem. It also spawns KOSAgentSystemManager. The AgentSystem starts the Parent ThreadGroup to which Places(described below) are added. Hence at any instant, the ThreadGroup structure looks like a tree as shown below:



Various Threads running under AgentSystem

As shown above, various ThreadGroups exist during the AgentSystem's execution. Agents present in a Place, is wrapped in an AgentWrapperThread(another subclass of ThreadGroup). Since ThreadGroup provides control over Threads present in a Group and Threads belonging to one group trying to manipulate Threads belonging to another group is subjected to scrutiny by installed KOSSecurityManager, Agents cannot manipulate threads belonging to some other groups. Because Places are implemented as ThreadGroup and Agents are embedded in AgentThreadWrapper, control is simple.

KOSPlace

KOSPlace class which implements a Place, is implemented as ThreadGroup. Whenever a new Place is spawned, the KOSPlace(subclass of ThreadGroup) is added to the Parent ThreadGroup started by the KOSAgentSystem. The KOSPlace along with providing interfaces to finding out services provided, mobility, it also provides interfaces for subscribing/unsubscribing to necessary channels present on the MessageBus. The KOSAgentSystemManager , after receiving an Agent stream from a client, makes an instance and asks the KOSPlace to insert it. The KOSPlace maintains a list of Agents currently present in its control. This list is used to provide status information, satisfy Disposal/Retract request.

KOSAgentSystemManager

The KOSAgentSystemManager provides GUI termed AdministratorControlPanel, through which an User can manipulate the properties of the AgentSystem, view the status of the system. If the user feels more Places have to be started, he does through the GUI. It also provides methods for starting and stopping of services. Apart from providing UI to the User, the KOSAgentSystemManager is in charge of inserting incoming Agents or Native Agents. All the queries fielded by GateManager is forwarded to AgentSystemManager which gets the result from KOSPlace.

KOSSecurityManager

The important class in the system, KOSSecurityManager, provides the assurance that Agents are prevented from doing actions which are potentially dangerous. At present, Agents are not allowed to open sockets(not even to their home, though this will be relented later), cannot read/write files, does not have access to System event queue etc. As is the principle in Java, all methods(like opening Sockets , reading/writing files) as a rule check with installed security manager whether the calling(invoking) thread has access to do such operation. Hence the KOSSecurityManager(a subclass of SecurityManager) is always checked whether an action is permitted or not. If the caller is Agent then immediately an Exception is thrown, thus preventing the code from doing anything which can be dangerous. When certificates, public key processing are supported, the API's for requesting capabilities will be provided, then an Agent can use these API's to request for more permission, which the KOSSecurityManager provides after authentication, thus certain privileged agents can open sockets or read/write files.

agentClassLoader

This class is a subclass of ClassLoader. An instance of this class is created whenever the System receives an Agent stream. The instance is provided with Kctp URL from where the Jar file containing the classes can be downloaded, immediately the agentClassLoader contacts the Agent's Home system and communicates with ObjectManager using Kctp protocol and downloads the Jar file, and loads the class and thus helping in creating an instance of the Agent. Once instantiated and inserted in a Place, whenever the Agent needs any class the Java design makes sure that the

ClassLoader that loaded this Agent is requested to load the newly asked class. The agentClassLoader has cached the Jar file locally, hence requests for any custom classes present in the Jar file can be fulfilled, if the class can not be found, the Primordial ClassLoader is asked to load the class in usual way. Since it is possible that the Agent may be asking for Java core classes or some classes from KOSAgent package, which the agent should not have access to, the agentClassLoader proceeds cautiously. Certain packages of Java core or KOSAgent core or other packages available may be restricted or even forbidden. For example if the agent asks for a class *java.lang.SuperPower*, if the loader after finding that the Primordial Loader is not able to find it, just loads it from the downloaded Jar, then this class, since it declares to belong to *java.lang* package, will get access to package-level accessible methods and fields present in various other classes in the package *java.lang*, which can lead to security breech. The agentClassLoader takes the following steps:

1. It checks whether the class is available in its internal cache, if present it is returned, or else it proceeds to the next step.
2. It then checks whether the requested class belongs to any of the Forbidden packages, if so an Exception is thrown, if not it proceeds to next step.
3. It then checks whether the requested class belongs to any of the packages to which access is restricted, if so it asks the Primordial loader to find the class. If Primordial Loader fails, the agentClassLoader immediately throws an exception signalling that an attempt to load non-existent class from a package to which access is restricted. If the Primordial loader finds it, the class will be returned. If the class does not belong to any of the restricted packages, it proceeds to next step, if Primordial loader fails to find it.
4. It looks for the requested class in the downloaded Jar, if found it is returned, if not it throws an Exception signalling that the requested Class was not found.

Thus the agentClassLoader ensures that classes that declare themselves as part of Forbidden packages are never loaded and classes that declare themselves as part of Restricted packages are loaded from local system only. All other classes are loaded in custom way, that is from Jars downloaded by Kctp. The list of packages restricted or forbidden is specified as properties(discussed later).

MessageBus

This class follows singleton pattern, only one instance of this class can be created. It allows Channels and Casters to be created. It provides interfaces for objects to register /unregister themselves from Channels. The implementation is similar to Java's Event delegation model. Channels allows objects interested in ChannelEvents(carriers of Messages published in Channels), to provide Listener Objects with specific interface, which will be called whenever any message is cast in the channel . At present Channels can be created only by System(AgentSystem) objects, in later releases privileged Agents will be allowed to create their own Channels.

ObjectManager

This class opens a Server Socket at port 9127 and waits for clients to connect and request for resources. It uses Kctp (discussed later) for serving classes. At present it serves classes to any client who makes request, if the asked for resource is present. At present it serves Jars, though the functionality to serve specific classes from Jars is available in the present implementation, it is not used(discussed later under Kctp).

GateManager

This class opens a Server Socket at port 8127 and waits for clients to connect make requests like, request for entry into the system ,request for status of some agent, request to dispose an agent and request to retract an agent or request to return an agent. This spawns a clientHandler which parses the request(Katp) and services the request. At present there is no restriction on clients who can request.

ImmigrationManager

This class takes care of mobility of an Agent, it makes requests to other AgentSystems for entry, return etc. on behalf of an Agent. This uses Katp for communicating with GateManager.

AgentThreadWrapper

This class, a subclass of ThreadGroup, acts as a wrapper for the Agent, after agent has been instantiated, and its Place assigned, this class spawns a thread which calls onDocking() method of the Agent for it to initialise , then it calls the run() method of the Agent to allow it to execute. Once the Agent stops executing, the control returns to thread embedded in AgentThreadWrapper, hence the Agent can be packed up and sent back home or put into persistent storage for later collecting by its Owner.

Katp

This is an Application-level, simple protocol meant for making requests/responses for providing Mobility to Agents. The present version of the protocol does not provide any Authentication facility. The protocol contains HTML like tags and value pairs for requesting and responding. The following list gives the tags that are used(Current Version is 0.9 - it is not yet Stable):

<KATP/0.9>

Tag which specifies the version of Protocol being used.

<ENTRY>

Tag which requests entry into the System.

<AGENT= filename >

Tag which specifies the Serialized file name of the Agent, which gives the name of the Agent. The convention followed is that the Agent when serialized, it is put in a file named as follows:

<agent-class name>.ser . For example for an Agent whose class name is myAgents.SpecialAgents.Messenger, the serialized file will be named as myAgents.SpecialAgents.Messenger .ser

<SIZE= stream size>

Tag which specifies the size of the Serialized file. This is similar to content-length. This tag is used in Request packet for making Entry request and also in Response packet , after Return Request has been granted.

<DISPOSE= agent-ID>

Tag which requests to Dispose an Agent whose ID is agent-ID.

<RETRACT= agent-ID>

This tag requests that agent specified by agent-ID to be retracted.

<RETURN= agent-ID>

This tag requests that agent specified by agent-ID wants to return back to AgentSystem.

<QSTATUS= agent-ID>

This tag requests to know the status of the Agent whose ID is agent-ID.

<RSTATUS= agent-status>

This tag responds with the status of agent queried by QSTATUS tag.

<SEND>

This tag responds signalling that the client who made request for entry or return can start sending the Serialized stream.

<OK>

This tag responds signalling that Disposal request was carried out.

<ERROR= error-index>

This tag responds that an error occurred, the error-index gives an index into the array carrying all the error messages related to Katp.

The Request Packet and Response structure is as follows:

Request Packet = {

<KATP/0.9>(

<ENTRY><AGENT= filename><SIZE= size>|

<QSTATUS= agent-ID>|

<DISPOSE= agent-ID>|

<RETRACT= agent-ID>|

<RETURN= agent-ID>

)

}

Response Packet = {

<KATP/0.9>(

<SEND>|

<RSTATUS= agent-status> |

<OK>|

<SIZE= size>

)

}

The URL looks like:

katp://<hostname>:<portnum>/

eg. katp://agentservices.com:8127/

All Request/Response packets starts with Protocol version tag, this lets the other peer whether it can handle the packet or not. The Error Messages are:

1. Packet not in correct form.
2. Protocol not understood.
3. No such Agent present
4. Agent ID is not recognised.
5. No services available now.
6. System is being shutdown.
7. Agent Entry failed.
8. Agent does not belong here.

At present Authentication of the clients is not done, later Certificate exchanges and query on list of services available and request to create an Agent on the AgentSystem on behalf of the client will be added.

Kctp

This Application-Level protocol, much simpler than Katp, is used for requesting for classes(served as jars). At present the protocol does not provide any Authentication facility. Though, as mentioned earlier, functionality to serve individual classes from Jar files is present, it is not used, because some experimentation is to be done to check

whether it is efficient to download each class separately or as whole. The following list gives the tags that are used(Current Version is 0.9 - it is not Stable):

<KCTP/0.9>
This tag specifies the version of Protocol being used.

<RESOURCE= jar-name>
This tag requests for a Jar file as specified by the name jar-name.

<SIZE= size>
This tag specifies that the requested resource is of size as specified, and that the client can start reading the stream.

<OK>
This tag specifies that resource has been successfully read.

<ERROR= error-index>
This tag specifies that an error as indicated by error-index has occurred.

The Request and Response structure is as follows:

```
Request Packet = {
                    <KCTP/0.9><RESOURCE= jar-name>
                 }

Response Packet = {
                    <KCTP/0.9>(
                                <SIZE= size>|
                                <OK>|
                                <ERROR= error-index>
                              )
                  }
```

The error messages are:

1. Protocol Not Understood
2. Packet not in correct format
3. Resource not found
4. Resource not accessible
5. System being shutdown

The URL looks like:

kctp://<hostname>:<portnum>/path/jar-name

eg. Kctp://agentservices.com:9127/users/raptor/Messenger.jar

Request for individual classes from a jar file is made in the same way Anchor is embedded in HTML pages. To request for a class by name Wanted.class from wanderer.jar, the request looks like:

kctp://agentservices.com:9127/users/raptor/wanderer.jar#Wanted.class

Properties

Various properties read from a file named *kosys.prop* dictates the behaviour of the AgentSystem and also tells various other information as described below.

kos.home

This property specifies the Home directory in which KOSAgent is installed.This is set as environment variable KOS_HOME.

Kosys.authority

This property specifies the User name to whom this AgentSystem and hence the Agents created by this system belongs.

o*bjman.root*

This property specifies the directory which ObjectManager is to use as root directory for looking for Jars requested. For example if resource requested is

/users/raptor/wanderer.jar

Then ObjectManager appends looks for users/raptor/wanderer.jar under the directory specified in *objman.root* property.

tmp.serbase

This property specifies the directory under which incoming and outgoing serialized Foreign Agents are to be placed.

tmp.home.serbase

This property specifies the directory under which incoming and outgoing Native Agents are to be placed.

init.places

This property specifies the number of Places that are to be spawned at AgentSystem start up.

max.place.count

This property specifies the maximum number of Places that the AgentSystem can spawn.

max.agent.count

This property specifies the maximum number of Agents that are allowed in a Place.

java.protocol.handler.pkgs

This property is used by Java core to look for packages to handle custom protocol. The value of this property specifies various packages into which the Java

VM has to look for to handle custom protocols. This is to be specified so that the Java runtime can find classes for handling Kctp and Katp

package.forbidden.<package-name>

This boolean valued property specifies whether a package is forbidden or not. If the property is not specified then it is taken as that the package is not forbidden. If a package SystemAdministration.passwordUtils package is to be forbidden to classes not loaded by Primordial Class Loader, then the property is specified as

package.forbidden.SystemAdministration.passwordUtils=yes

package.restricted.<package-name>

This boolean valued property similar to the one discussed above, specifies whether the package <package-name> is restricted or not.

Conclusion

Summary

Various existing Mobile Agent Systems were studied and some experience was gained with few of them. This resulted in the survey presented in the report. Based on the experience the Architecture was drawn up. Then a part of the functionality mentioned in the KOS JAF was implemented. The release is used for gaining more experience and experimenting with security and to make decisions about how to handle various exceptional cases.

Applications

The Architecture drawn up is meant for a generic Mobile Agent System (for any kind service providing), which can be deployed on the Internet to provide services to the public(with security increased). Some of the Applications where this System can be used is:

- Mobile Agent System based Search/Mining Engines, which provide a Agent System layer on top of the Database(resource) for Agents to visit and do various customised search locally.
- Mobile Agent System based Information providing Services, like Share Market Quote Services, where Agents can visit and register with Channels of their choice and wait for updates to happen so that they can act accordingly.

Future Work

Presently implemented System has AgentSystems individually providing services with individual GateMangers. This system is perfect for individual users. The AgentSystem constructed can also be termed as Personal AgentSystem(PAS). Future work will have services implemented on CORBA ORB that will allow various AgentSystems to co-operate with each other, and have a single GateManager acting as Firewall for the whole group of AgentSystems. Since the services for inter-AgentSystem communication will be implemented on top of CORBA ORB, various other AgentSystems(not necessarily KOSAgent systems) with proper bridges can work with GateManager.

Since CORBA will be used for internal communication, various services provided at some AgentSystems, can be availed from another AgentSystem belonging to the same network controlled by AgentHost. Legacy applications can be plugged into AgentSystems as Services, through ORB. Using an ORB, helps in controlling Regions, thus enabling load balancing across various AgentSystems belonging to same Authority. When each AgentSystem starts up they register themselves and various

services provided by them with GateManager, thus enabling the GateManager to provide the clients, with information about services provided by AgentSystems.

Digital Identities, Certificates will be used for Authentication. Thus improving the security provided by the AgentSystem, and also allowing certain privileged Agents to do certain actions which are not possible without proper Authentication. Capabilities API will be added which the Agents can use to request more privileges from AgentSystems for performing certain actions, which are not normally allowed.

The present protocol used will be improved with more functionality, like requesting for list of services available, list of AgentSystems available, and also requesting to create an Agent on behalf of an client.

At present Agent history is not maintained, this will be added later. The addition of Agent History will help an AgentSystem to know whether the Agent visited any untrusted AgentSystems before visiting this system, thus treat such Agents with caution.

At present inter-agent communication is not provided, once ORB service interfaces are defined, communication between Agents will be implemented as an ORB service, thus care will be taken that an Agent does not harm another Agent(in a simpler case, Denial Of Service). Some kind of Agent Communication Language may be developed to experiment with in enabling Agents communicating knowledge between themselves.

References

1. **Mobile Agents - William R. Cockayne** & **Michael Zyda** (Manning)
2. **Developing Intelligent Agents For Distributed Systems : Exploring Architecture, Technologies and Applications- Michael Knapick & Jay Johnson** (Computing McGraw-Hill)
3. **Inside CORBA - Thomas J. Mowbray** & **William A. Ruh** (Addison Wesley)
4. **Client/Server Programming with Java and CORBA - Robert Orfali** & **Dan Harkey**
5. **Aglets** related information can be obtained from ***http://www.trl.ibm.co.jp/aglets/***
6. **Odyssey** information can be obtained from ***http://www.genmagic.com/***
7. **AgentTcl** related information can be obtained from ***http://www.cs.dartmouth.edu/***
8. **Mole** related information can be obtained from ***http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.html***
9. Many papers related to Mobile Agents can be found at ***http://www.informatic.uni-stuttgart.de/ipvr/vs/projekte/mole/***
10. **LiveAgentPro** related information can be obtained from ***http://www.agentsoft.com***
11. **KQML, KIF** related information and links can be found at ***http://www.cs.umbc.edu/agents/***
12. **Voyager,** an ORB which facilitates Mobile Agents deployment, related information can be found ***http://www.objectspace.com***
13. **CORBA** related information can be found at **OMG** site ***http://www.omg.org***
14. **Agent Society Home** ***http://www.agent.org/***
15. **KASBAH** related information can be had from ***http://kasbah.media.mit.edu/***
16. **ARA** related information can be found at ***http://www.uni-kl.de/AG-Nehmer/Projekte/Ara/index_e.html/***