Mobile Objects and Agents (MOA)

Architectural Specification

Dejan Milojicic, William LaForge and Deepika Chauhan

The Open Group Research Institute
11 Cambridge Center
Cambridge, MA 02142
[d.milojicic, d.chauhan, b.laforge]@opengroup.org

Version 1.2

1 Introduction

This paper presents the system architecture of the Mobile Objects and Agents (MOA) system. Mobile agents are programming entities that can freely roam the networks and perform the tasks on behalf of the agent owner. An owner can be the user who created the task, or another user if ownership can be delegated.

In this document we define the basic system components and describe their interactions. Each component will be identified, and explained within the context of the system as a whole. This document will not address application programming interfaces, nor classes required to implement these components. This is the first of three documents that will be provided for this project. The second one is functional specification, which will define interfaces and state associated with various objects comprising MOA, and describe basic abstractions in greater detail. The design specification will concentrate on the design details of mobile agents infrastructure.

1.1 Project Goals

- **Provide a standard and potentially widely accepted environment for mobile agents.** Solutions that require changes to standard or widely adopted solutions are not acceptable.
- **Efficient interagent communication support.** Our project is aimed at applications that can consist of multiple agents that communicate and cooperate while fulfilling their target goal. Therefore interagent communication is of high importance.
- **Effective agent naming schemes allowing agent tracking and manipulating.** Because of the wide variety of agent applications and consequently the variety in agent travelling patterns, the flexible schemes must be provided to track an agent's current location and to support the control of agents.

- **Simple design and implementation.** Rather than aim at complex algorithms for agent tracking, we prefer a set of configurable schemes, adaptable to different application needs.
- **Extensive resource management.** Agent resources, such as open channels, time-to-live, number of allowed clones, etc. are tracked and limited by user set defaults.
- Component model. MOA is conforms to the JavaBeans component model which allows agent applications to be stored, retrieved, inspected and re-used.

1.2 Project Constraints

- **Minimize changes to the underlying development Java environment.** While changes to the Java VM can provide for some additional characteristics, such as transparent checkpointing of execution point, departing from standard solution can significantly impact deployment of agents.
- **Compatibility with the "off-the-shelf" tools**, such as web browsers and web servers. Similar to the previous constraint, departing from standard tools will limit the agent deployment and usability.
- **We shall not research and develop security.** We are seriously considering security issues but no specific work will be conducted within the scope of this particular project. An attempt will be made, nevertheless, to avoid imposing any incompatibility with later security extensions. All available existing security support will be used, but no additional work will be invested in security.
- **Limited interoperability with other languages** (e.g. as CORBA does) **and/or other agent systems** (e.g. as Aglet system does). At a later stage of the project, our interfaces may conform to OMG MAF specification. This should not be a major effort, and will certainly be well isolated from the existing work. Such interoperability is not a short term goal of the project.

1.3 The Agent System supports the object model of agents and places that agents visit. The hardware/software platforms supported by MOA are describe in Section 3 and 4 respectively. The object model is described in Section 4. The Agent System and Agent Services are described in Section 5 and Section 6 respectively.

2 Hardware Components

This project will be independent of the underlying hardware environment as much as possible. The target environment consists of x-86 PC's and Java environment. However, most of the development will be conducted on Windows NT, on PC compatible computers. We shall attempt to use computers geographically distributed across continents. We shall conduct the bulk of the development on local area networks in order to hasten development, but most of the final experiments and development work will be conducted on the Internet.

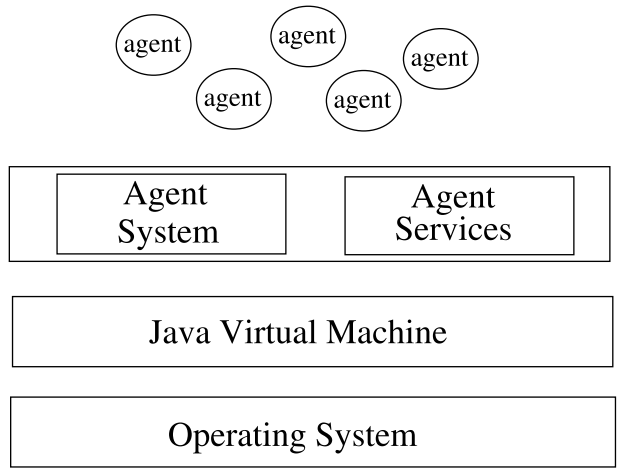

FIGURE 1. High Level Organization of the Mobile Agent Software Components.

3 Software Components

In this section the software components and the dependencies and relationships between them will be described. The main software component comprising agent support is the Agent System. Agent Services is an additional module that provides extended support for agents. Figure 1 describes the high level organization of the mobile Agent System, Agent Services and other modules in the system. Agent applications are executed on top of the Agent System. There can be multiple Agent Systems on a node, and multiple applications (agents) can be executed in an Agent System. Only agent applications are mobile and they can roam among the Agent Systems that are stationary to designated nodes. The applications, Agent System and Agent Services are written in Java. Therefore they require Java virtual machine to execute on every node an agent visits. We are currently using JDK 1.1 which also includes various packages, such as objects serialization, reflection, Java Beans and Remote Method Invocation, which we briefly describe in the following text.

All this functionality match pretty well with the current and future needs of mobile agents. Object serialization takes care of transforming objects in a transportable form, so that they can be transferred across the network and then deserialized into its original form. Object serialization is necessary for mobile agents implementation today.

Reflection API supports introspection of the classes and objects in the Java Virtual Machine. It will be useful in future applications when agents are stored and retrieved at will, and where they can come and return at arbitrary times; in such a setting, reflection will allow inspecting the agents and its objects and classes and making knowledgeable decisions and changes to agents at the run-time. Java Beans relies on reflection extensively.

Java Beans has the potential to facilitate the agents to become just any other component on the Internet: the component that can be re-used, stored, moved around, inspected upon arrival and based on that accepted or refused, etc. The MOA system has employed software component technology as a basis for its development. Most modules in MOA are designed as Java Beans so as to give them the abovementioned characteristics.

Remote Method Invocation (RMI) facilitates remote method invocation and the passing of objects across the network. In its first version, the agent system will not rely on the RMI, however, there is nothing to prevent the agent applications to use RMI (there is a requirement to establish and clean connections across the migration, which is the responsibility of the application using RMI).

4 Object Model

The agent object model defines agent abstractions, such as agents, places, primitives for inter-agent communication, and inter-agent synchronization and naming. In addition, it defines and implements methods available to control agents and other abstractions. This section will briefly introduce the basic agent-related abstractions. A more detailed description of each abstraction will be provided in the functional specification. The object model is implemented by the Agent System (described in the following section).

An **agent** is a basic abstraction described in this project. Agents in this project are written by application programmers, typically in Java (although there already exist some interfaces between other languages and Java, this project will exclusively address agents written in Java). The underlying support for agents is provided by the Agent System. Agents consist of constant code and variable data they accumulate as they migrate between the nodes. Agents typically invoke a "*move*" method (also called "*go*", as in the Telescript model, or "*migrate*" as in classical process migration) autonomously, i.e. by interpreting their application code. Once an agent visits a new node, its code is executed from the beginning. If there is a need for restarting an agent at a particular execution point, this is achieved by maintaining state in the application, in the form of switch statement, similar to the methodology deployed in Aglets [Agle96]. This is described in further detail in next section.

Places are abstractions representing locations that agents visit. An agent can only visit a node if a place has been set up for it. While places are introduced for different purposes, the major reasons are security and resource management. Agents should only be allowed access to resources of the place they are visiting at a particular moment. Allowing all agents to visit a single place, and thereby allowing access to the same resources, would open security holes. Having multiple places provides for finer granularity in resource allocation and access.

Before allowing an agent to arrive at an agent system, a negotiation takes place. Negotiation deals with security and resources availability, it is a verification dialogue conducted between an agent and a place.

Communication Channels may be opened between agents and or places in order to facilitate communication between two agents, or between two places, or between an agent and a place. Agent application can synchronize their execution by meeting at a place, or by exchanging messages. Agent communication and cooperation are one of the focuses of this project. There exist many other projects dealing with mobile agents, but none of them stresses agent cooperation. We believe that mobility and dynamics required in communication requires additional support, currently nonexistent in Java. This is one of the focuses of this project.

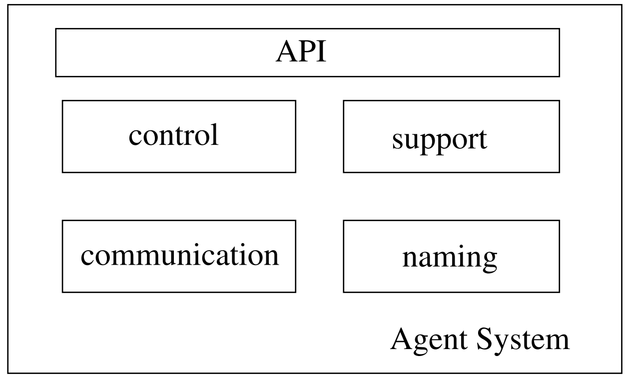

FIGURE 2. The Relationships between the Mobile Agent Software Components.

Names of agents and places is performed using Internet naming schemes. Places are based on agent names and their current location. Agent names are based on the place of their birth (resembles the last name of a person), and on a local identifier (resembles the first name of a person).

5 Agent System Architecture

The Agent System provides the basic support for mobile agents. It supports the following functionality: API; control; naming; support; and communication. The architecture of an Agent System is presented in Figure 2.

API subsystem consists of the following modules

- *Agent* is the basic abstraction in MOA that models the application semantics as well as supports mobility of code and data. Agents may move from node to node with unlimited frequency whilst maintaining open channels of communication between agents across migrations. Agents may also be cloned by an agent application.
- *Place* is a container for security and resource management. An agent always resides at a place and an agent may leave the place behind on migration. Communication channels may be attached to a place to permit data to be transferred to/from the place. A place also serves in providing a finer granularity for resource management.
- *AgentProperties* specifies the properties that characterize an agent, e.g., name, family name, home agent environment, alternate home agent environment, etc. The agent cannot directly modify the properties.

Control subsystem consists of the following modules:

- *Monitor* creates the user object when requested by the MoaApplet. The monitor collects information of interest to a user object from other sites. The information consists of traffic volume, message counts, and selected logs.
- *AC* is a core module that is responsible for managing the services and resources requested by the agent/place of the MOA system. This includes channel requests, logging services, move requests, cloning, snapshotting etc.

- *AcTakedownMgr* maintains a list of agents at a site and brings an agent down when its time-to-live has expired. There is one instance per agent system. On arrival the agent is registered with the AcTakedownMgr. The time-to-live for the agent is determined based on the LocalPolicy for the agent.
- *AcChannelMgr* supports the opening and closing of communication channels between agents and places. The AcChannelMgr is the central hub to which all actions on communication channels are directed.
- *AgentPolicy* specifies the limits on the resources that are available to an agent, e.g., maximum number of channels that may be opened by an agent, maximum number of threads that may be requested by an agent, maximum number of clones that may be spawned by an agent, etc.
- *LocalPolicy* is the result of arbitration between the AgentPolicy and the HostPolicy. The arbitration involves reconciliation of resources that are requested by the agent versus resources that a host is willing to make available to the agent.
- *Family* the logical grouping to which an agent and its clones may belong. As clones spawn off additional clones they are distinguished by the generation number. Thus an agent and multiple generations of its clones belong to the same agent family.

Communication subsystem accepts incoming communication from remote nodes and sends outgoing messages. It is responsible for maintaining communication channels for cooperating agents. Maintaining communication channels after migration is also achieved by using the naming facility, as the name of an agent helps in locating new agent location, which is further used for reopening agent communication channels. The communication module can achieve this in a more flexible fashion. This support is added by means of an interposing layer between the standard communication methods provided by the Java run-time and the application. This layer maintains sufficient amount of state in order to allow the checkpointing and restoration of the communication state across agent migration

- *Net* supports connections between software components located on local and remote java virtual machines. Connections are made by specifying the name of the component and the host and port number.
- *Messenger* supports two-way messages between components on local and remote java virtual machines. The messages are sent by specifying the component name and host and port number.

Support subsystem consists of the following modules

- *Mover* is the mechanism by which the agent state which has been checkpointed is moved to the destination node. A move of an agent from the source node to the target nodes involves cooperation between the movers on the target and the source nodes. The move operation is also accompanied by negotiation with the negotiator prior to a successful completion of a move operation.
- *Logger* supports capturing and selective displaying of information about the agent and related abstractions activities. The following information about agent is logged: places visited, communication, meeting, forking, etc. Similarly, it is possible to log information about places: the agents that visited the place, that met at a place, the communication that took place, etc. It is possible selectively to inspect logged information. Because of the large amounts of data, not all information is logged, nor is it

logged for all agents. It is possible to specify which data and for which agents and/or places will be logged, both in a static and dynamic manner.

- *Killer* takes down the Java Virtual Machine after a specified number of elapsed milliseconds. The This capability is useful in testing.

Naming subsystem consists of the following modules

Nameserver is responsible for resolving the names of agents and places (locations they visit). The name of an agent environment is used as a target to move an agent to. Additionally, a place can also be used as a destination. Places can be used for improved inter-agent communication. Agents communicating at the same node can do so, much faster than if they are communicating between two different nodes. Names of agents are used for subsequent accessing/searching/controlling of agents.

Beside resolving the agent address, the naming module also takes care of locating agents across their migration. Simple location schemes are provided, such as searching, updating, registering and forwarding. It will be possible to dynamically deploy specific procedures, suitable for given types of applications.

6 Architecture of Agent Services

Agent Services enhance the Agent System by supporting control of agents, interaction with WWW tools, and additional auditing, debugging and monitoring support for agents. The Agent System is able to function without Agent Services, in a limited fashion. Agent Services consists of the following elements.

Front end is responsible for interaction with the MOA system. It consists of MoaApplet, BatchDriver, and Graphical User Interface (GUI). *MoaApplet* provides the connection between the operator and the user object. At startup, the applet sends a request to the monitor to create the user object. It also supports the launching of pre-configured agents as well as execution of commands. The *BatchDriver*, a batch script, is the command line interface that enables the operator to interact with the moa system. The BatchDriver supports operator logon, agent launching, display of launched and pre-configured agents, etc. In addition, MOA provides an operator with a *GUI* to the moa system. The operator is able to logon, launch and agent, display lists of pre-configured and launched agents, display and update the agent family, debug using MOA's trace facility. In addition, the operator may also execute commands targeted at an agent.

User serves as an interface to the front end, tracking information of interest to a user, allowing the user to run a user applet repeatedly without the need to re-enter user-specific data. There is one user object for each operator (human). The user object is preconfigured with a set of properties, e.g., operator name, password, path, etc. The properties are set via a moabatch file (.mb extension). In addition, the user object tracks the agents, sites and other information of interest to the operator, using the monitor (control subsystem) to gather this information from remote sites.

Home Service serves primarily as a respository for snapshots and logs of all agents, for which that site serves as a home or an alt-home (alternate home). In addition it also serves as a placeholder for properties of agents at that site. The home service must track which users may access data for which agents as well as which agents may use that site for home service. The home service also supports snapshot retrieval queries The hom-

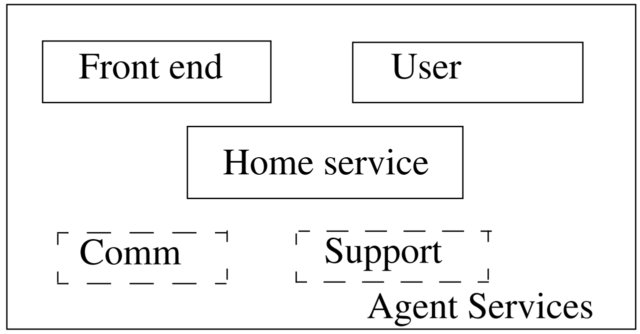

FIGURE 3. The Relationships between the Components of an Agent Services

service also provides interactive access to agents and places via the Monitor. Agents can be created, killed, suspended, resumed, searched for, etc. Places can be inspected for residing agents.

Agent services rely heavily on the communication and support subsystems (shown in broken lines) of the agent system. For example, a log is received by the home service as a log message, thereupon validated by the homeservice, and subsequently forwarded to the logger for storage. Similarly, a snapshot is received as a message object by the homeservice, validated by the homeservice, and then cached by the homeservice.

Agent services use the services of the communication and support subsystems of the agent system, discussed in section 5. Agent Services and its components are presented in Figure 3.

7 Mobile Agents and the WWW

Mobile agents will be integrated with the WWW. In the initial phase, while Internet security is still in its infancy, this integration will be limited. As security improves, a higher degree of integration will be reached. There are good signs that this will happen in the near future. Digital signatures are already being integrated with Java. Jeeves server and servlets are starting to be deployed on the server side, in a similar manner to applets on the client side. All of this represents a convenient infrastructure for wider deployment of agents.

Integration with the WWW also involves integration with web browsers in order to allow agent invocation. At the moment, due to security restrictions, it is not possible to provide a complete Agent System within the WWW browser. In order to overcome security limitations, agents run within an Agent System.

As already discussed, agents also conform to WWW naming conventions. Places, the objects that agents visit while roaming the network, are named using *host* and *port* (a subset of URL). Similarly, the agent names are derived from the originating node addresses extended with local identifiers. Naming will be designed and implemented in a configurable manner in order to leave opportunities for adoption of other naming schemes, as WWW naming evolves. One opportunity is URNs, but it is questionable when this standard will (if at all) be adopted.

8 Related Work

This section describes different agent implementations that are known in literature or those that are already available as products.

Java based agents

The **Aglets** project, conducted by IBM Japan, supports agents written in Java [Aglets]. It aims at standardization of agent facility.

IBM Aglets, along with contributions from this project, General Magic, Crystalit and GMD, have been proposed to OMG as a submission to a request for proposal for Common Facilities [OMG95]. Aglets represent a Java implementation of a standard agent transfer protocol and agent facility.

The **Mole** project at University of Stutgart was one of the first academic efforts in providing mobile agents in Java [Mole]. It has undergone a couple of phases. In the most recent phase, Mole relies on Java and RMI. The Mole project also doesn't support checkpointing execution point. Mole is used by Siemens for experimenting with mobile agents and AI, as well as by some other industrial sponsors, such as Daimler-Benz and Tandem.

Sumatra is a Java based agent project at the University of Maryland [Rang96]. Sumatra provides support for execution point checkpointing. However, Sumatra requires changes to the Java virtual machine. This is a compatibility issue that our project is not willing to undertake. Sumatra uses agents for optimization of communication between different nodes. The benefits are demonstrated experimentally. Distributed resource monitors are also provided [Rang96a].

CyberAgents is a commercial product developed by FTP Software, Inc [Cyberagents]. It is a software package that supports mobile agents for various tasks, such as system administration, virus checking, platform inventory information collection, etc. CyberAgents support sophisticated user interface and standard control of agents, but with limited inter-agent communication and collaboration.

Tcl-Tk based Agents

Agent Tcl project is one of the first to provide agents written in Tcl-Tk [AgentTcl]. It provides rich semantics of agents and agent interaction. The code is publicly available from their web site. Agent Tcl is intended for mobile computers, handling temporarily disconnected computers. Agent Tcl also deals with navigation and adaptation. It uses dedicated Agent RPC (ARPC) for agent intercommunication.

Ara started as a Tcl-Tk based mobile agent project, at University of Kaiserslautern [Ara]. Recently it has also moved towards Java, claiming multiple language support. Ara developers are currently addressing transparent execution checkpoint, similar to Sumatra.

Tacoma project started as a collaboration between University of Tromso at Norway and Cornell University, USA [Tacoma]. It supports rich semantics of agents and agent interaction. Recently they claim multiple language support. Cornell's part of the project deals with fault tolerance and security. Their work is also related to ISIS technology.

Other Languages

Telescript, developed by General Magic, is the first well known system that supports agents as a commercial product [Telescript]. Originally, Telescript was dedicated to closed networks. Since Java's appearance, this market has declined and General Magic has refocused Telescript on the Internet, but with less chance for success. Our project plans to adopt many concepts from Telescript, such as the concepts for agents, places, "*go*", etc. General Magic also sports a system classed Odyssey, and have now moved towards a java implementation of agent technology.

Python [Python], developed by Guido van Rossum, is being used as a base for mobile agents, under the name Knowbots [Hylt]. Knowbots are targeted for applications consisting of multiple cooperating agents that travel and communicate among themselves.

Obliq programming language [Card95] is used to support so called migratory applications [Bhar95]. Migratory applications are well suited for mobile computing. They can checkpoint not only their state, but also the environment that was run at one site and then restore it on another.

Alternative Sources of Information

An additional source of information about existing agent project can be found in a list of people attending Dartmouth Workshop on Transportable Agents [Dartmouth].

An agent mailing list regularly posts agent related news. An archive is available from [Agent News].

A thorough survey of process migration, including a small section on mobile objects and agents can be found at [PM].

Acknowledgments

David Black and Shai Guday read this document and provided feedback that improved the contents and presentation.

Bibliography

[Agent News] Agent mailing list archive available from: *http://www.cs.umbc.edu/agents/agentnews/*.

[Aglets] Aglets home page: *http://www.ibm.co.jp/trl/projects/aglets/*.

[Ara] Ara Home Page: *http://www.uni-kl.de/AG-Nehmer/Ara/*.

[Bhar95] Bharat, K. A. and Cardelli, L. "Migratory Applications", Proceedings of the Eight Annual ACM Symposium on User Interface Software Technology, November 1995.

[Card95] Cardelli, L. "A Language with Distributed Scope". Proceedings of the 22nd Annual ACM Symposium on the Principles of Programming Languages, pages 286–297, 1995.

[Cyberagents] CyberAgents Home Page: *http://www.ftp.com/cyberagents/*.

[Dartmouth] List of Invitees/Bio/Projects from Dartmouth Workshop on Transportable Agents: *http://www.cs.dartmouth.edu/~agent/workshop/people.html.*

[Jeeves] Sun Jeeves home page: *http://java.sun.com/products/jeeves/.*

[Mole] Mole Home Page: *http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole.html.*

[OMG95] OMG, *Common Facilities RFP 3*, OMG TC Document 95-11-3, November 3, 1995.

[OMG96] OMG Submission, *Mobile Agent Facility Specification*, IBM, Opengroup, General Magic, Crystalit, GMD Focus, OMG TC Document cf/96-08-01, August 15, 1996.

[OS] Java™ Object Serialization Specification, Revision 0.95, available as: *http://chatsubo.javasoft.com/current/serial/index.html.*

[Python] Python home page: *http://www.python.org/.*

[PM] Process Migration: http://www.opengroup.org/~dejan/papers/indx.htm.

[Rang96] Ranganathan, M., Acharaya, A., Sharma, S., Saltz, J., “Network-aware Mobile Programs”, to be presented at the Usenix 1997 Conference.

[Rang96a] Ranganathan, M., Acharaya, A., Saltz, J., “Distributed Resource Monitors for Mobile Objects”, submitted to IWOOS96.

[RMI] Java™ Remote Method Invocation Specification, Revision 0.9, available as: *http://chatsubo.javasoft.com/current/doc/rmi-spec/rmiTOC.doc.html.*

[Tacoma] Tacoma Home Page, *http://www.cs.uit.no/DOS/Tacoma/.*

[AgentTcl] Tcl Agent Home Page: *http://www.cs.dartmouth.edu/~agent/.*

[Telescript] Telescript Technology: Mobile Agents, General Magic White Paper, available as *http://www.genmagic.com/Telescript/Whitepapers/wp4/whitepaper-4.html.*