An analytical survey of mobile agent for resource management in a network

F. J. Ogwu
M. Talib

Department of Computer Science
University of Botswana
Botswana

G. A. Aderounmu

Department of Computer Science & Engineering
Obafemi Awolowo University
Ile-Ife
Nigeria

Abstract

Management in today’s enterprise networks requires new paradigms for coping with the heterogeneity, distribution, dynamic nature, and complexity of the managed resources. Traditional mobile scripts and RPC-based management approaches are not sufficient in this context. Hence a more dynamic and robust approach based on intelligent mobile agents is proposed framework for resource management using intelligent mobile agents on an enterprise network.

After a brief introduction of agents, a special attention is paid to itinerant agents and mobile agent technology. The proposed framework is discussed and a simulation is done for network resource management on an enterprise wide network to demonstrate the working of the framework.

Keywords: Mobile-agent, itinerant-agent, visiting-agent, agent-code, agent-thread, transfer agent.

1. Introduction

The complexity of today’s enterprise networks is becoming a new challenge for management engineers. New management paradigms are required to deal with the heterogeneity, complexity, distribution, and
the dynamic nature of the resources in the global, open environment. In the traditional approaches of network and service management, i.e. in the field of SNMP and TMN-based management, an agent has a standardized or proprietary interface, providing a fixed functionality, and is running in a specific place. This provides a limited view of the local resources at a remotely accessible management interface [1]. This is used by management applications to monitor the state of the resources or to perform configuration operations via the SNMP or the CMIP protocol. For several reasons such approaches are insufficient in a dynamically changing, nondeterministic, heterogeneous environment that obtains in network and service management area today. The current management standards are not flexible enough to build dynamically extensible management solutions. To reduce requirement on human expertise and network resources in installing, maintaining and deploying the management expertise, management solutions in the form of autonomous, Intelligent mobile entities will play an important role [1].

Intelligent mobile agent approach to resource management on an enterprise network provides a more versatile, and robust solution to management problems than these conventional management schemes. These agents are pieces of software that act on another entity’s behalf (where the “entity” can be either a person or another software program) and which possesses certain autonomy [2]. This entity is referred to as the agent’s owner. The agent owner may require the agent to have several orthogonal agent properties apart from the mandatory properties of agents depending on the use to which the agent is to be put. Some key agent properties are described below:

**Autonomy** is a property of agents which delegate authority to the agent to act based on its own accord. The more autonomous an agent is the more its actions are unpredictable. Agent autonomy is usually constrained by its owner’s restrictions, several other factors like the protocols of communication and social behavior in a multi-agent community also restrict agent autonomy.

**Communication** is a property of agents that allows them to exchange instructions and data with other agent.

**Intelligence** refers to the agent’s ability to make decisions based on its previous experiences. This leads to the concept of agent rationality. A rational agent, should do whatever action is expected to minimize its performance measure (minimize its cost function) on the basis of the
An agent needs to be able to learn.

*Learning* refers to an agent’s ability to take whatever information it has gained about the environment in which it is deployed and use it to modify its own behavior. By building on that information an agent will, at least, not repeat a mistake, at best it will develop new more useful behaviors [4].

*Reactivity* is the ability of an agent to sense changes in its environment and act according to those changes.

*Adaptability* is a property that makes an agent adapt to changes in its environments. A control agent for example may need to adapt to changes in its feedback signals by filtering superimposed noise.

Agents are also *Goal-driven*, that is, proactive and temporally continuous, that is, they execute continuously.

*Mobility* is the ability of an agent to start its “execution at one place, then to move to another location (taking with it both code and data) where it resumes its execution [2].

An agent is different from an object of object-oriented programming in that an agent is an active entity that has the following properties:

- It is situated within an execution environment.
- Possesses the following mandatory properties: Reactivity, Autonomy, Goal-driven, and Temporally Continuous.
- It may possess any of the following orthogonal properties Communicative, Mobility, Learning and Believable.

A framework that can provide mobile functionality for agents is discussed.

Mobile agents are pieces of software that have the ability to start their execution on one computer, suspend execution and move to another computer through the network, resume execution, and so on.

2. **Mobile agents**

Mobility as described above is an orthogonal property of agents — that is, not all agents are mobile. In terms of ability to move, an agent can be described as *stationary or mobile.*
A stationary agent executes only on the system where it begins execution. If it needs information that is not on that system or needs to interact with an agent on a different system, it typically uses a communication mechanism such as Remote Procedure Calling (RPC). A mobile agent is not bound to the system where it begins execution. The ability to travel allows a mobile agent to move to a system that contains an object with which the agent wants to interact and then take advantage of being in the same host or network as the object. The autonomy of mobile agents essentially consists of space autonomy that is an agent can determine to travel to any place in the network; with or without further instructions from its original location.

A mobile agent is different from a mobile script: a mobile script executes remotely but cannot return to its original place and it is not autonomous. Mobile scripts are pieces of software, written in a scripting language that can be executed remotely. Examples of mobile scripts are Java applets and Extensible Markup Language (XML) agents. Mobile-agent systems differ from process-migration systems in that the agents move when they choose, typically through a “jump” or “go” statement, whereas in a process-migration system the system decides when and where to move the running process (typically to balance CPU load) [5].

Mobility is essential in management of network traffic. Assume a particular scenario when a database is to be searched for a record on a remote system in a computer network, the records would have to be passed to the client system individually during the search thereby greatly increasing the network traffic. Another situation also arises when using Remote Procedure Call (RPC) in a client-server system. The network has to carry $2N$ number of messages for $N$ services requested for by the client, the RPC system consisting of two messages; request and response for each service rendered. Such network traffic increase is unwarranted, and the situation can be improved by simply deploying a mobile agent to accomplish such tasks. The agent can gather the instructions about the requested service and move to the server system where it carries out the instructions and accesses the services locally; the agent then carries only the “results” back to the client site if need be. This system requires less use of network bandwidth, and there is no need to maintain a physical link between the client and the server throughout the operation: the link is only required during the agent’s travel. Also the client and the server do not need to agree on a particular communication protocol once the client agent understands the interface provided by the server.
3. **Mobility implementations**

An agent system is a platform that has the ability to create, name, execute, dispatch and terminate agents [2].

A *place* is a context within an agent system, in which an agent can execute. A agent system can support more than one place: many agents can execute at the same place, and agents can travel to different places located on different agent systems.

The agent transfer is worked out by serializing and un-serializing the agent code. The agent code is transformed into a stream, and vice versa as shown in Figure 1.

![Figure 1](image)

**Figure 1**
Transformation of agent codes into streams

The itinerary pattern is an autonomous decision of the agent. It has the responsibility of determining either statically or dynamically its navigation pattern. When the agent wants to go on a trip it determines its destination and other activities it needs to carry out on the destination system, saves its state while suspending its execution. The agent then serializes its state and code. *Serialization* is the process of creating a persistent representation of the agent object that can be transported over a network [6]. The serialized agent is then encoded and transported with
the help of the agent system and the operating system using the available transport protocol over the network.

There is a small piece of memory resident code in the agent system that monitors the presence of the agent code and does some verification on it before starting up the code.

When the agent code arrives at the destination, it is decoded and the persistent representation of the agent is deserialized (D-S). The agent class is instantiated, and the transferred agent is restored. The monitor creates a new thread of execution for this agent. The code for implementing this mobile framework is written in C++. A C++ class has been developed to provide the mobile features that the agent can inherit. The protocol of communication with the agent system has been handled in this class and the necessary interface where the agent can determine its movement is published and could be used easily in the agent code. The agent system with an itinerant agent is shown in Figure 2. The system only has a single agent executing at any instant in time; the agent after transferring its code terminates its execution as explained earlier. The agent system on the source and destination engines makes sure that the agent is not lost during its itinerary.

![Figure 2](image)

Agent system showing an itinerant agent
The interface to this agent system (Figure 3) provides a mechanism for monitoring agent activities on the current node; the system could also be configured from this interface.

![Agent system](image)

**Figure 3**
Agent system as an interface between agent and host platform

As described above the agent system has several components and functions it performs. The agent system could be seen as a large distributed system that has components running in host machines. The host machine hardware and operating system comprises the engine on which the agent system runs on. Host machines that have these agent system components running on them are therefore prepared for agent activities, and hence can be visited by the itinerant agents. The agent system components allow the agent to access the host resources it needs for computation, communication, and navigation. It also ensures that the host system remains secure while the agent is active. The relationship between the agent, the agent system, and the host is shown in Figure 3. The agent system translates communications between the agent and the host, so that the host can compute and communicate for the agent, and help it steer its way through the computer system. The host machine houses the resources that the agent needs to interact with.

4. **Agent life cycles**

The agent starts its life as a persistent and inactive object in a storage medium on a host system until the agent system activates it. It then starts performing its various activities; a accessing the resources it is expected
to use. The agent may then decide to move to another host to continue its activities or to access a resource to get the necessary information that would help it to continue its activities. The agent then informs the agent system of its intended voyage and necessary connection is made with the destination system. The agent is serialized again into a persistent object that encapsulates its state and codes. This persistent form is then moved to the destination system where it can resume its activities. This process is repeated over and over again on different hosts throughout the lifetime of the agent.

5. The agent system monitor

The agent system monitor provides a graphical user interface for the user to observe the agent activities going on his machine. The agent activities are represented graphically and the results gathered on the local machine and other hosts are represented. The monitor also provides an interface for configuring the agent system; the agent could be informed of more host machines available on the network that the agent could visit, the agent could also be restricted from visiting certain machines and other tasks may be added or removed from the agent activities. The agent system monitor is shown in Figure 4.

![Figure 4](image_url)

Mobile agent system monitor

The agent system waits for an agent to move to its local system. Whenever an agent comes into its system it does some security verification on the agent and if the agent passes these security tests a thread of execution is created for the agent. Whenever the agent is done and informs the system of its intended itinerary it sets up a communication with
the remote agent system and prepares the way for the agent travel. After
the agent has completed its movement to the remote destination the
system does some house cleanup job and prepares itself for another incom-
ing agent. A flowchart of the agent system activities is shown in Figure 5.

Where:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Start</td>
</tr>
<tr>
<td>Inputs</td>
<td>Wait and accept user inputs</td>
</tr>
<tr>
<td>AG</td>
<td>Is there a new agent in the system?</td>
</tr>
<tr>
<td>DAC</td>
<td>Destroy agent code</td>
</tr>
<tr>
<td>PV</td>
<td>Passed verification?</td>
</tr>
<tr>
<td>CAT</td>
<td>Create agent thread</td>
</tr>
<tr>
<td>AR</td>
<td>Agent ready to leave?</td>
</tr>
<tr>
<td>SC</td>
<td>Set up communication with remote system and transfer agent</td>
</tr>
</tbody>
</table>

Figure 5
Agent system activities
6. Resources management

The mobile agent was used to demonstrate mobile ability of agents and the ability to transparently help in network resource management. The agent monitored segments on an enterprise network for network traffic. The agent then uses this information to configure machines on this network transparently to places of less traffic. In a segmented network, certain segments may have very high traffic whereas others may be relatively unused. Such segments with very high traffic may experience lower throughputs e.g. contention based networks or the network may become generally slower e.g. token rings. The agent moves around the enterprise wide network and reconfigures machines to segments of lower traffics; the user does not know about the reconfiguration but may only notice the general performance improvement because of the reconfiguration. Figure 6 shows an algorithm the agent uses to reconfigure machines on the network.

![Diagram showing the algorithm for machine configuration on a segmented network.](image)

**Figure 6**

*Algorithm for machine configuration on a segmented network*

The agent also does some other tasks of gathering information about user machines like available storage space on mass storage media, resource availability like printer.

Availability etc: this information is then passed to other machines on the network to inform users of resource availability.
Such result could be used in distributed operating systems for transparent resource allocation.

7. Model of dynamic route decision for itinerant agent

The path of an itinerant agent is determined by certain constraints that pertain to its functions. If the agent performs a regular service to all nodes on the managed network its path may be a predetermined one going in a cycle from one machine, around the network and back to that machine in a regular order. In such a model the path determination is static and could be supplied to the agent by the agent’s manager. On the other hand path determination may be dynamic and is based on agent decision at a particular instance depending on the demands placed on the agent for certain agent services.

The model for the dynamic route determination is defined as a three-tuple \( \langle A, P, M \rangle \) where:

- \( A = \{a_1, a_2, a_3, \ldots, a_n\} \) is the set of \( n \) possible locations the agent can get to.
- \( P = \{p_1, p_2, p_3, \ldots, p_n\} \) is the probability vector that specifies the probability of a location being selected as the destination of the location.
- \( M = \{m_1, m_2, m_3, \ldots, m_n\} \) is an oldness vector that keeps track of the last time a location was visited.

At the beginning of agent activities all entries in the probability vector is set to \( 1/n \). This distributes the probability of being visited equally among the locations. A location \( a_k \) with highest probability value is then selected as the destination location.

For every change of location by the agent the oldness vector is modified as follows:

\[
m_k(t) = 0, \\
m_i(t) = m(t - 1) + 1 \quad \text{for every } i \neq k.
\]

Where \( t \) represents the last time the location \( a_k \) was visited.

The probability vector is afterwards computed as follows

\[
p_i(t + 1) = p_i(t) - 1/N \quad \text{for } p_i(t) > 0
\]

and \( N \) is a positive integer,

\[
p_k(t + 1) = 1 - \sum p_i(t + 1) \quad \text{for } i \neq k
\]

this ensures that all nodes are visited evenly based on the agent’s determination, the oldness vector is used by the agent for other administrative purposes.
8. Security issues

A major concern specific to mobile agents is the protection of the servers running the agents. Running arbitrary programs on a machine is dangerous: a hostile program could destroy the hard drive, steal data, or do all sorts of other undesirable things. This risk must be thoroughly addressed if mobile agent environments are to succeed.

The complement of server security is agent security: whether the agent can trust the server on which it is executing. A mobile agent might contain secret information such as proprietary data and algorithms. Worse, servers might have an incentive to subvert the computation of a visiting agent.

It is possible now to deploy a mobile-agent system that adequately protects a machine against malicious agents [7]. Numerous challenges remain, however:

1. protecting the machines without artificially limiting agent access rights;
2. protecting an agent from malicious machines; and
3. protecting groups of machines that are not under single administrative control.

An inadequate solution to any of these three problems will severely limit the use of mobile agents in a truly open environment such as the Internet. Fortunately, groups are now exploring many new techniques, each of which address (or partially address) one of the three problems (e.g., agents paying for resources usage with electronic cash [8] which allows them to live and propagate only as long as their cash supply holds out). Although many technical advances (and user-education efforts) must be made before these three problems are solved adequately for all Internet applications, current work is promising enough that, within a few years, mobile-agent systems will be secure enough for many applications.

9. Conclusions

Mobile agents are an effective choice for many applications, for several reasons [9], including improvements in latency and bandwidth of client-server applications and reducing vulnerability to, network disconnection. Many applications will find mobile agents the most effective implementation technique for all or part of their tasks. The simulation
program to demonstrate this mobile agent paradigm gave satisfactory results. The performance analysis and source code descriptions are discussed offline.

References


[14] www.meitca.com/HSL/Projects/Concordia

Received November, 2005