MOBILE AGENTS

Term paper.
Prepared by Tatsiana Levina

Abstract: The exponential growth of the information has made intelligent software assistants increasingly important. There are two types of intelligent agent:

-- stationary agents
-- mobile agents (MA)

A stationary agent executes only on the system on which it begins execution. If it needs information not on that system or needs to interact with an agent on another system, it typically uses a communication mechanism (e.g., remote procedure calling).

A mobile agent is not bound to the system on which it begins execution. It is free to travel among the hosts in the network. Created in one execution environment, it can transport its state and code with it to another execution environment in the network, where it resumes execution. In this paper I will focus on mobile agents.
1. Classification of Software Agents

According to Gilbert et al.\(^6\), software agents can be classified in terms of a space defined by the three dimensions of intelligence, agency and mobility (figure 1).

![Figure 1: Scope of intelligent agents (based on Ref.\(^6\),\(^9\))](image)

**Intelligence** is the degree of reasoning and learning behavior. Intelligent agents can be classified according to their capabilities to express preferences, emotions and beliefs and according to their ability to fulfill task by interpreting, learning and planning techniques.\(^3,5,9,11\)

**Agency** is the degree of autonomy and authority vested in the agent and can be measured by the nature of the interaction between the agent and other entities of the system (e.g. negotiation skills). At a minimum, an agent must run asynchronously. The degree of agency is enhanced if an agent represents a user in some way. A more advanced agent can interact with data, applications, services or other agents. According
to their capabilities, agents are called autonomous, collaborative, cooperative or negotiating agents.

*Mobility* is the degree to which agents themselves can travel through the network.\(^3\)\(^,\)\(^5\)\(^,\)\(^9\) There are different types of paradigms which support communication between entities in a distributed computer system.

--*Message passing* was the earliest paradigm proposed, which transports data and allows processes communicate by sending and receiving messages explicitly.

--A *remote procedure call* (RPC) is a higher level communication paradigm. With RPC a process passes not only the parameters of the called procedure but also the code of the procedure itself.

--Whilst RPC allows only "code mobility", the concept of a mobile agent supports "process mobility", i.e. a program execution can migrate in a computer network. A mobile agent carries to the destination not only code but also the state information of the agent. An agent's state consists of data set (includes agent's global variables and instance variables) and execution state (includes local variables and the active threats). Migration can be strong and weak. With strong migration the whole agent state is captured by underlying system and transferred with the code to the next location. As soon as the agent reaches its new location, its state is automatically restored. Strong migration can be a very time consuming (the agent state can be large) and expensive operation, therefore a weak migration was introduced (with it only the data state information is transferred).\(^9\)

Degrees of mobility are summarized in the Table 1.

<table>
<thead>
<tr>
<th>Migration Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong Migration</td>
<td>Migration code + data + state</td>
</tr>
<tr>
<td>Weak Migration</td>
<td>Migration code + data</td>
</tr>
<tr>
<td>RPC</td>
<td>Transport of code + data</td>
</tr>
<tr>
<td>Message Passing</td>
<td>Transport of data</td>
</tr>
</tbody>
</table>

*Table 1: Degrees of mobility*
2. Definition of Mobile Agents

A mobile agent is an agent which is "not bound to the system where it begins execution. It has the unique ability to transport itself from one system in a network to another. 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 to take advantage of being in the same host or network as the object." 7

3. Mobile Agent Architecture

This section describes general architecture of a mobile agent. It includes:

a suitcase, a briefing, a mobile agent server, a hop instruction, and the agent itself.

A suitcase is the piece of data that the MA carries with it as it moves from site to site. This is the long-term memory of the MA that can include a list of all sites the MA visited, tasks performed at each site, and the results of these tasks.

A briefing is the data that a MA receives as it enters a site. This can include any advice and site-dependent information of which a MA might need knowledge such as local file systems and databases. The agent server acts as the monitor at the entry gate for an agent when migrating to a new system.

The agent server controls the security and authentication of the agent, briefs the agent as it enters the system, and executes the agent code 2, 4. The agent server is sometimes referred to as the object manager, and it is this unit of the MA system that absolutely controls the MA environment 10.

The MA uses the hop instruction to move from one site to the next. The hop instruction uses as parameters an agent server, the code of an agent, and a suitcase.
4. Examples of MA systems

As per Rothermel et al. 9 I distinguish MA systems based on language implementation. Hence, I recognize the following types of MAs:

1. low level language approaches,
2. Java based approaches,
3. other high level language approaches and
4. language independent approaches.

Low Level Language Approaches

The lowest level at which a mobile agent system can be implemented is the network protocol layer. Networked approaches provide only basic functionality for mobile agents; higher order functionality, like security issues, have to be implemented on top of the agent system.

One of the examples of mobile agents is given by the Network Command Language is an earlier implementation of this type of approach.

o An interface language consists of a generic set of low level interface routines.

o Each server which supported this interface language could be accessed from clients locating on a variety of machines running different operating systems.

o A high-level and machine dependent RPC from a client could be broken down into the low-level and machine independent routines of the interface language.

o The resulting agent program could then be transferred to the server, where its execution would recreate the original remote procedure call. 9

Java Based Systems

The Aglets system was developed by Lange et al. at the IBM Tokyo Research Laboratory.
- The message passing scheme supports synchronous and asynchronous communication between agents.
- Aglet proxies allow a uniform and location transparent access to other aglets independent of the actual context of the aglet.
- A whiteboard mechanism allows agents to subscribe to message types and to broadcast messages to all subscribed agents.
- Each agent inherits a travel itinerary, that specifies travel patterns and automatic failure handling.
- Agent mobility is implemented by weak migration, and can be issued by the agent itself (push) or by an agent in the destination context (pull).
  
**Concordia** is a Java-based mobile-agent system with a strong focus on security and reliability.
- The system moves the agent objects code and data, but not thread state, from one machine to another.
- *Concordia* agents are bundled with an itinerary of places to visit, which can be adjusted by the agent while en route.
- Agents, events, and messages can be queued, if the remote site is not currently reachable.
- Agents are saved to a persistent store, before departing a site and after arriving at a new site, to avoid agent loss in the event of a machine crash.
- Supports weak migration.
- Agents are protected from outside influence through encryption while they are in transmission or stored on disk
- Agent hosts are protected from malicious agents through cryptographic authentication of the agent's owner, and access control lists that guard each resource.

**Other High Level Language Based Systems**

*Telescript* is a language designed for mobile agent programming by White at General Magic Inc.
- It is an object-oriented class-based language.
An agent process uses strong migration to get from one place to another.

All processes and further objects are managed by an engine (a virtual machine running in a system process of the underlying operating system).

The engine is essentially a *Telescript* interpreter with a collection of built-in classes for creating, transmitting and manipulating agent processes.

A *Telescript* agent can communicate with other agents in two ways:

- it can meet with an agent that is in the same place; the two agents receive references to each other's objects and then invoke each other's methods; and
- it can connect to an object in a different place; the two agents then pass objects along the connection.

It is one of the most secure, fault-tolerant and efficient mobile agent systems.

It has been withdrawn from the market, mainly because it was overwhelmed by the rapid spread of Java.

The *MESSENGERS* system developed by Bic, Fukuda and Dillencourt from the University of California, Irvine.

It supports applications structured as collections of autonomous objects, called *Messengers*.

*Messenger* scripts are written in a C and are compiled into a form of byte code for more efficient transport and parsing.

A messenger script is a sequence of statements, which can be of one of the following types:

- Computational statements enable the *Messenger* to perform arbitrary computations. They include all standard C assignment and control statements, involving arbitrary variables and constants;
- Navigational statements provide the *Messenger* with mobility, permitting it to create and destroy logical nodes and/or links, and to move within the logical network.
- Function invocation statements provide an interface to the system’s environment.

Agent mobility in this system is implemented by strong migration.
Agents build an application-specific logical network used for navigation.

The system supports fault-tolerance by automatic snapshots and restarts.\textsuperscript{9,13}

\textbf{D'Agents} is a mobile agent system developed by Gray et al. at the Dartmouth College, Hanover.

- \textit{D'Agents} can be written in multiple languages (Tcl, Java and Scheme).
- Agents can communicate with each other regardless of implementation language.
- \textit{D'Agents} implements strong (Tcl and Java) and weak (Scheme) migrations.
- \textit{D'Agents} only provides low level and typeless communication mechanisms to send and receive single messages or message streams.
- The \textit{D'Agents} architecture has four levels.
  - The lowest level is an application programming interface (API) for the available transport mechanisms. Currently only the TCP/IP protocol is supported (can be extended to other transport mechanisms, like SMTP).
  - The second level is a server that runs at each network site to which agents can be sent.
  - The third level consists of one interpreter for each language supported by the system. Each interpreter has four components:
    - the interpreter itself,
    - a security module that prevents agents from taking malicious action,
    - a state module that captures and restores the internal state of an executing agent and
    - a language-specific wrapper for the generic API.
  - The top level of the \textit{D'Agents} architecture consists of the agents themselves.\textsuperscript{3,9,13}

\textbf{Language-Independent Approaches}

\textit{ARA} (Agents for Remote Action) has been developed by Peine and Stolpmann at the University of Kaiserslautern.
System core implemented in native code (C), applications agents implemented in various languages (C/C++, Tcl)

ARA implements strong migration

ARA provides only a basic client-server style of communication between agents.

Higher order communication mechanisms must be built on top of these basic concepts.

ARA is a multi-threaded system (i.e. when a new agent arrives, ARA simply begins execution in a new thread, and when one agent wants to communicate with another, it just transfers the message structure to the target agent, instead of using inter process communication).

Each agent in ARA system is equipped with resource accounts called allowances (in order to limit the resources used by an agent). The creator of an agent specifies a global allowance which is valid for the whole lifetime of the agent. Upon entering a place, the allowance can be further restricted (by the agent or by the place) to a local allowance.

Agents can share allowances or transfer resources between their accounts.\textsuperscript{3, 9, 13}

5. Benefits and Applications of Mobile Agents

Now that the foundation of MAs has been established focus will be directed towards questioning if MAs are really needed.

The bottom line of this debate is that RPC's can accomplish equivalent results as a MA. RPC's accomplish the goals of a MA by mixing Client/Server interactions by using message passing that involves simple datagrams and socket connections. So what are the benefits of MAs?\textsuperscript{5}

- Reduce communication costs: Mobile agents may reduce communication with respect to latency, bandwidth and connection time. Communication latency can be reduced if agent is send across the network with a sequence of service requests rather than issuing each server request by a separate RPC. MA also offers a solution for real-
time systems, where network latencies are unacceptable, since they can be send from a central controller to act locally (figure 2). 7, 8, 9

![Figure 2. Mobile Agents reduce network load (adopted from 7).](image)

- **Dynamic Protocols**: A computer usually supports a limited number of protocols. If some protocols are missing in order to access, view or process data, the protocol must be downloaded manually. MAs allow dynamic protocols, i.e. new protocols to be installed automatically as needed. 9

- **Asynchronous and autonomous computing**: It is possible to imbed needed tasks into mobile agents and then dispatch them into the network. After being send out to the network, the MA becomes independent of the process that created it and can operate asynchronously and autonomously, so that you can reconnect later to collect the agent (figure 3). 7, 8, 9

![Figure 3. MAs allow disconnected operations (adopted from 7).](image)
A flexible distributed computing architecture: mobile agents provide a unique distributed computing architecture which functions differently from the static set-ups. It provides for an innovative way of doing distributed computation.

Currently there are no specific mobile agents applications, but there are applications that benefit from the mobile agent paradigm:

--E-commerce: Here MA technology can help to solve the problem of real time access to remote resources.

--Personal assistance: The ability of MA to execute on remote hosts makes them appropriate as a personal assistants, so that MAs will perform tasks on the network on behalf of their creators.7, 8

--Distributed information retrieval: Let say, for example, we want to create search indexes. Instead of moving large amounts of data to the search engine MAs can remote information sources where they locally create search indexes and than send them back to the system of origin.

Among other possible applications for MAs are: telecommunication networks services, workflow applications and groupware, monitoring and notification, and parallel processing.7, 8

6. Drawbacks of the MAs

However, MAs have some limitations. First, since the agent's complete code is sent out every time it moves to a new node in the network, the size of the code must be minimized in order not to compromise network bandwidth and transfer speed. Similarly, the size of the MA's data suitcase must be minimized as well since its contents are also sent along with the agent code when the agent moves. As more data is collected in the suitcase, the larger the agent grows. Again, this could affect network congestion and transfer speed if a suitcase becomes excessively large. Moreover, in order for an agent to move to a different node, the agent must pack itself, unpack itself, and be compiled. The
time and processor overhead from the actions is another factor that needs consideration. Among the drawbacks of the agent approach is the increased complexity of the security threats. In order to avoid a mobile agent acting like a virus or a worm a security infrastructure that provides protection to agents as well as to agent places has to be establish. This is especially important in an open, dynamically changing computing environment. Next subsection will discuss security of MAs in more details.

6.1 Security of Mobile Agents

Just like any distributed system, mobile agents are subject to the following, well-known general security threats:

- unauthorized access to information,
- unauthorized manipulation of information,
- unauthorized access to services,
- vandalism (damaging the system without benefit).

In agent systems, these threats appear in various forms and at different locations. In many cases these threats are enlarged through the aspects of mobile code and autonomous activities.

Threats for Agents

The behavior and state of an agent may be exposed or modified by changing the internal code or data. Agents code and data can be readable for the agent place if no encryption is used. If the agent carries data in encrypted form, it would need a decryption key to handle the data by itself; but then the agent place would also have access to this secret key.

Zapf et al give an example of the "owner of an agent that was sent to bargain for some goods and that carries electronic money in some form may be subject to attempts to manipulate its buying decisions in favor of another, malicious user. Like-wise, the communication activities of a buying agent may be recorded and replayed, causing severe liability problems for the owner of the agent."
Threats for Agent Places

Agent places may be threatened by the execution of malicious agents in several ways:
– Agents may gain an unauthorized access to facilities of the place.
– Agents may gain access to internal data of places and other agents at the place.
– Agents may block the execution of other agents by an intensive utilization of places and services (denial of service attack). Here agents may choose the one of the following forms of attack:
   – Spamming: A place will be weaken or damaged by an artificially high resource utilization.
   – Spoofing: An agent tries to adopt a false identity and to gain unauthorized access to data.
   – Viruses, worms, Trojan horses: Agents could transport and implant malicious code for software attacks.

The agent place must protect the following system resources: the operating system, file system, memory, and access to other programs. In order to protect the system resources the place should be able to identify an agent, therefore authentication of the agent owner (user) through use of an access control mechanism is necessary.  

7. Conclusion

Most researches agree that mobile agent technology offers many advantages for distributed applications. Mobility and autonomy, for example, make permanent connections unnecessary. Hence, agents are well suited for coping with expensive network links or links that may be temporarily unavailable. The usage of agents can also be appropriate in situations where large volumes of data have to be shipped over the network while the processing code itself is rather small. Another application arena for mobile agents is electronic commerce and electronic information services. Agents can roam the electronic market place in order to search for desired information or to conduct trading activities.
Nonetheless, the existing research has failed to provide data to support the idea that MA's can accomplish the same task faster than alternative methods. Moreover, Chess et al. say "with one rather narrow exception, there is nothing that can be done with mobile agents that cannot also be done with other means" (the exception is remote real-time control). Furthermore, most research papers on MA's fail to mention the average size of the agent's code. Can the average agent's size impose unwanted network congestion during transmission? In addition, agents have to be packed, unpacked, and compiled when an agent moves to a new node. How much processor overhead will this incur?

To become more attractive, MA technology has to overcome problems related to connectivity (e.g. access through firewalls), location transparency and security.

To summarize, although the theoretical advantages of MA's can make MA's to appear as a better choice, the above mentioned limitations/questions might prove MA's a poor choice in practice.

Reference:

1. Aridor, Y. and Oshima, M., "Infrastructure for Mobile Agents: Requirements and Design"(8)
4. Caperton, C., "Intelligent Agents: Evolution and their Nomadic Notions".(2)


8. Lange, D. B. and Oshima, M., "Seven Good Reasons for Mobile Agents"(7)

9. Rothermel, K. and Schwehm, M., "Mobile Agents"(6)


11. Vulcan, N., "Economic Implications of Agent technology and E-commerce"(11)

12. Zapf, M., Muller, H., Geihs K., "Security Requirements for Mobile Agents in Electronic Markets" (4)

13. The Mobile Agent List

http://www.informatik.uni-stuttgart.de/ipvr/vs/projekte/mole/mal/preview/preview.html