RC 19887 (88465) (12/21/94) Declassified 3/16/95
Subject Area: Computer Science/Mathematics 23 pages

IBM Research Report

Mobile Agents: Are They a Good Idea?

David Chess, Colin Harrison, Aaron Kershenbaum
IBM Research Division

T.J. Watson Research Center
Yorktown Heights, New York

IBM Research Division
Almaden · T.J. Watson · Tokyo · Zurich

1 Abstract

Mobile agents are programs, typically written in a script language, which may be dispatched from a client computer and transported to a remote server computer for execution. Several authors have suggested that mobile agents offer an important new method of performing transactions and information retrieval in networks. Other writers have pointed out, however, that mobile agent introduce severe concerns for security. We consider the advantages offered by mobile agents and assess them against alternate methods of achieving the same function. We conclude that, while the individual advantages of agents do not represent an overwhelming motivation for their adoption, the creation of a pervasive agent framework facilitates a very large number of network services and applications.

2 Introduction

The idea of performing client-server computing by the transmission of executable programs between clients and servers has been popularized in recent years by researchers and developers interested in intelligent network services, most notably by White & Miller at General Magic, Inc. [23], but also by the developers of TCL [17]. Mobile agent-based computing may be viewed as an extension of well-known methods of remote dispatch of script programs [10]. or remote submission of batch jobs [6]. The most significant of the extensions lie in the area of security, since an important goal of this work is to enable spontaneous electronic commerce; that is commerce which does not require the prior conclusion of a trading contract between the two parties. Security is in fact a significant concern with mobile agent-based computing, as a server receiving a mobile agent for execution may require strong assurances about the agent's intentions.

These security concerns have led us to a critical examination of the use of mobile agents in network services, to determine whether the benefits they offer compensate for the concerns that they raise. In this paper we examine various arguments that have been adduced in favor of mobile agents, comparing the individual benefits they claim with alternative methods of achieving the same result, and also considering the overall benefit of a mobile agent framework for network services. This assessment considers both technical and commercial factors. We do not consider the question of the desirability or necessity of competing with General Magic or other vendors of mobile agent technology.

We begin by a description of the attributes of mobile agents and then proceed to analyze the pros and cons of the individual claims (trees) and the aggregate merit of an agent framework (forest).

3 Mobile agent-based computing

The mobile agent concept is illustrated in figure 1. A client computer consists of an application environment, for example, OS/2 or Microsoft Windows, which contains one or more applications for interaction with a remote server. These applications may include information searching and retrieval, transaction front-ends, or mail clients.

These applications are bound to an execution environment for mobile agents. Via the APIs, the application can pass parameters to various classes (not necessarily object-oriented classes) of agent programs, and likewise the agent programs can return parameters to the application programs. These classes may be part of the basic agent execution environment, agents distributed with the OS/2 or Windows applications or agents received by the client from a server or other peer on the network. In principle there may be no application program, the agent programs can themselves perform presentation on the client device's user interface and collect information directly from a keyboard or other input device; in this case the agent programs - or the agent execution environment - - must bind to the user interface libraries of the client device. The agent execution environment will also need to bind to various operating system functions, such as the memory manager, the timer, the file system and so forth. In particular the agent execution environment needs to bind to the message transport service in order to send and receive mobile agents agents via the communication infrastructure.

Figure 1: Conceptual model for mobile agent computing

When an application needs to send mail or perform a transaction, it will assemble the required information and then pass this via the API into the agent execution environment. This will initiate the execution of an instance of a particular class of agent as a process within the agent execution environment. This may correspond to an operating system process or an operating system thread or it may be managed by a threads package within the agent execution environment. The agent execution environment will have access to many different agent programs, which provide different services to the client applications. For example, one may act as a delivery agent for electronic mail, another may deliver a database retrieval request to a server, submit the request and return the result to the client application, yet another may navigate its way among multiple servers, asking each in turn for updates on a particular topic. The agent program may be built from procedural components or from classes of objects. In either case, the agent has bindings to functions within the agent execution environment, including functions imported from the operating system, the application or other sub-systems, as well as other agent programs.
The program may be executed in either machine language or an interpreted (virtual

machine) language. In order to support heterogeneity, it is often preferable to express the agent in an interpreted language. There is a performance penalty for this, but since most of the agent processing is done not in the agent itself, but rather in the functions to which it binds, this may be acceptable. Interpreted languages also have the advantage of late binding; this enables the agent to contain references to functions or classes not present on the system at which it is launched, but which are available at its destination. Interpreted languages are also easier to render secure than machine language, since the language developer explicitly controls what system resources are accessible. (Provided that gaping loopholes such as PEEK and POKE are rigorously excluded.)
The information assembled by the application is accepted by the agent as part of its initialization and at at certain point in its execution, the agent will execute an instruction which has the following effects:

- Either the current agent process is suspended in the agent execution environment, or a new agent daughter process is created.

- The suspended process or the new process, including its process state, stack, heap and all external references is collected and processed into a message expressed in a machine-independent form, for example Abstract Syntax Notation 1 [16]. This step is facilitated if the agent is built from object classes and in an interpreted language. In particular, if it is known that the identical classes are resident at the destination, the agent may be reduced to object references, instance data and process state data. If the agent is expressed in an interpreted language, the state data is captured on the stack and there is no need to save registers.

- The message may be addressed explicitly to a final destination, or it may be directly initially to a post office function which can perform address resolution, or to intermediate destinations, which route the agent on the basis of its content (Semantic Routing).

- The message is handed to message sub-system and routed directly or indirectly to the destination server, where it is delivered by the server's message subsystem to the agent execution environment.

- In the agent execution environment the received message is reconstituted into the executable and the process or thread is dispatched.

- Execution continues at the next instruction in the agent program.

This is effectively a process migration, but one that is performed for the purpose of moving the agent from a client which has a request - - for information, for a transaction, for mail delivery - to a server which is capable of satisfying the request. During execution at the server, the agent passes the information it received from the client application to server application functions and perhaps receives other information in return. At the completion of this stage, it might perform one or more of several functions:

- It might terminate its execution.

- It might simply suspend at the server, waiting for some event to be delivered from a server application. We would say that is has become a 'resident' agent at the server. Resident agents may become permanently resident if there is some repeated service desired by the user.

- It might repeat the migration progress, either by forking a new daughter process or by suspending and migrating itself. This second migration might return the agent to its originating client or it might continue to another server or another client.

In particular, the agent may be able to perform a recovery action and visit another server if the required service is not available or is otherwise unsatisfactory or (equivalently) the agent may be able to determine that it should also visit another server based on data it has received from the current server.

3.1 Security

There are several security issues to be considered in mobile agent-based computing:

- Authentication of the user, that is, the sender of the mobile agent, by the server, and authentication of the server or agent execution environment, to prevent the spoofing. (It is not at all clear how this latter function can be implemented, given that the agent is passive during the authentication process.) The server may wish to be able to authenticate the sending user uniquely or it may be satisfied to know simply that the user belongs to a group of authorized users. Some servers may not require any authentication all, if they have no protected information or functions. The authentication information may be conveyed by the agent itself or it might be transmitted separately, for example between authentication servers at the client and server. The outcome of the authentication processes is that the user/agent knows the identity of the server/agent execution environment and the server/agent execution environment knows the identity of the the user/agent. This authentication is based only on header information transmitted with the agent; the server still has no idea what the agent wants to do.

- Determination of whether the user has authorization to execute agents at the server and which functions may be used and determination of whether the agent will attempt to infect the server, deny service to other agents or otherwise attempt do harm to the server or other agents. The server's agent execution environment will re-constitute the agent into an executable. However, before the server dispatches the executable, it may wish to examine the agent code to see what resources it proposes to access. This may be part of a general access control function or it may be part of a virus immune system function. If the agent language supports self-modification (as does Telescript), this may be an insufficient test, since the as-received agent may during execution be able to transform itself from a benign to a malignant entity. Following successful completion of this test, the agent execution environment will then permit the agent access to server resources, depending on the privileges of the user.

- Determination of the agent's ability or willingness to pay for services provided by the server (unless these are free). During execution the agent acts autonomously on behalf of the user. Since the agent is consuming at least computational resources at the server, and may in fact be performing transactions for goods, the user also requires considerable assurance that his or her liability is limited. In the case of General Magic's agents, the Telescript [1] language provides a method of authentication transmitted with the agent and also a method whereby the agent carries with it a quantity of an electronic currency (Teleclicks). During execution of the agent by the server, the server is entitled to transfer currency units from the agent to the agent execution environment as a form of payment. The user's liability is limited to the quantity of currency which the agent was issued by the client. In Telescript execution environments, an agent which exhausts its currency is killed. However, the user also requires assurance that the agent execution environment cannot fake the quantity of currency transferred and that the server is indeed providing the contracted services.

3.2 Virus detection

Analysis of the agent itself, to determine whether it is likely to exhibit virus-like behavior is a difficult problem. It is difficult to define necessary and sufficient tests that the agent must pass in order to determine whether its intentions are benign or whether it intends to infect or otherwise corrupt the host system. It is not the case that virus detection is undecidable in (and only in) Turing-complete programming languages. Nor is it the case that it is possible to write a virus in any Turing-complete language. Turing completeness is really rather a red herring when thinking about viruses, because:

- It is easy to design a non-Turing-complete language in which a virus can be written (just include an "infect" verb somewhere in the set of primitives).

- It is easy to design a Turing-complete language in which no virus can be written (and in which, therefore, the virus-detection problem is easy; the answer is always: "No, that is not a virus"). Consider, for instance, a language with the full programming power of REXX, but able to do input only from the keyboard, output only to the screen, and with no access to any underlying operating system commands or functions. We could write arbitrarily-complex games or Eliza programs in it, but since programs written in it cannot read or write other programs, they cannot be viral. (In theory, you could write an entire virtual operating system, including a file system, in this language, and there could be virtual viruses within that system, but that is not of practical relevance.)

Turing completeness only comes in very slightly: if you have a language that includes the ability to implement the "spread" operation, and the language is Turing-complete, then Cohen [7]. has shown that perfect virus detection is impossible. But his result does not say anything one way or the other about systems that are not Turing-complete, or that do not make the "spread" operation possible.

[1]Telescript and Teleclicks are trademarks of General Magic, Inc.

This is important for the mobile-agent question in at least one very large way: it means that one could design a mobile-agent system in which agents are written in a Turing-complete language, and as long as the "spread" operation cannot be implemented (as long, that is, as agents cannot alter other programs), we can still avoid having viruses.
A simple example of this would be an agent language with the basic syntax of REXX (say), but with only a very limited set of powers:

- To alter its own internal state variables,
- To make database queries in the current server,
- To move to another server,
- To send text messages back to its owner.

Despite being Turing-complete, there is no way to write a virus in this system. You can even allow such agents to add and update database entries (under proper access controls, of course), and as long as nothing ever interprets the contents of a database entry as an agent, we still do not have virus problems.
Mobile agents are not the only method by which viruses might be propagated in network services, although the use of mobile agents may greatly facilitate their propagation. Nor are viruses the only epidemic threat to network services, other effects such as mail broadcast storms are at least as likely and equally hard to deal with. J. Kephart [15]. has been studying the propagation of viruses in networks and will soon complete an initial architecture for the detection and confinement of these and other abherent behaviors of network-based services.

3.3 Issues

The use of mobile agents appears to offer certain advantages for client-server computing, but as we have noted above, also raises some difficult issues:

- Efficiency: Does the agent execution environment require significant computational resource? Does the transmission of a transaction or other request via a mobile agent result in more or less network traffic than alternate methods?
- Flexibility: Can the use of mobile agents provide a more flexible and robust method of communication than alternate methods? Is it likely that agent execution environments would be rapidly deployed on network servers?
- Security: Is there a useful compromise between the desire to isolate the agent execution environment from the system and application functions and the need to provide access in order to accomplish the users' tasks? Is is possible to define a language such that it provides sufficient expression for client-server interaction, while being sufficiently restricted that the server and other agents cannot be compromised?

4 Alternatives to mobile agents

For completeness, we mention here briefly the alternatives to mobile agents for client-server interactions. The dominant methods are messaging [9]., simple datagrams [21], sockets [19], remote procedure call [5], and conversations [8]. The primary distinction among these is between asynchronous protocols, for example, messaging, and synchronous protocols, for example, RPC. Mobile agents employ messaging frameworks for transport, and hence are asynchronous. In the rest of this assessment, we will use the term *messaging* to characterize asynchronous client-server interactions and *RPC* for everything else. In both cases, the client and server exchange data which is to be processed by specific procedures at the remote CPU. Neither party specifies how the data are to be processed; each has implicit knowledge of the capabilities of the remote procedures. This contrasts with mobile agents, which communicate both data and their own procedures and which exploit procedures resident at the client or server.

The *Remote Procedure Call (RPC)* extends the traditional procedure call mechanism of pushing parameters, registers and a return address onto the stack and then performing a jump to the procedure's entry point. In the RPC case, the client and server open a communications channel between the client application and the server process. The RPC parameters are passed to an interface routine, which marshals them into a form suitable for transmission and they are then sent explicitly to the server process. The RPC packets are received by a corresponding interface routine, unpacked and passed to the server procedure. The procedure processes the parameters and (generally) produces a return RPC, which is transmitted back to the client process. Both parties must use a common interface definition (although heterogeneity of hardware and operating system software is possible). While a local procedure call can be executed in at most a few microseconds (not including the execution time of the procedure itself), the RPC introduces overhead due to marshalling, transmission, and unpacking and has a typical latency of a few milliseconds. Like the local procedure call, the RPC is synchronous; the client process suspends, maintaining the entire process state, until it receives the return RPC from the server. Secure RPCs add authentication and encryption facilities to the client-server communication, but introduce significant overhead [22].

Messaging is emerging as a popular alternative to RPC for client-server communication. It is an outgrowth of both electronic mail systems and earlier distributed computing schemes in which applications communicated via files or pipes. The client application composes a message, typically composed of tagged or structured text, which is to be delivered to an appropriate software processor for the type of message. Messaging systems may employ a message transport service provided by by an electronic mail service, for example, Message Queuing Series [1], Simple Mail Transport Protocol [18], Vendor Independent Messaging [2]. The required processor type is indicated in the message header. The message is generally addressed indirectly, that is, the client may not know the explicit network address or even the identity of the destination server. The resolution of addresses is performed by intermediate steps of processing, such as post offices.

Messaging is inherently asynchronous; once the client has handed off the message contents to the messaging sub-system, it continues execution. If in the future, the client receives a response message from the server, it must restore the application state in order to process the response. For example, if the user is engaged in a dialogue with a reservation service, several iterations may be required between the client travel planning application and the Computerized Reservation Service (CRS), before the user has identified a suitable flight and seat; during these iterations, the client and the CRS must both maintain transaction state until it is committed.

Because the communication is asynchronous, the latency in messaging is both higher and less predictable than in the RPC case. As a result messaging may be less effective for one-to-one communication than RPC, but for one-to-many communication, which is typical of servers in network services, the throughput may be higher, since the client process does not need to suspend while waiting for the response. As in the RPC case, secure messaging can offer authentication and encryption; there is an equivalent overhead, but because the process is asynchronous, the overhead is less burdensome.

The strength of RPC lies in its high efficiency and low latency. The strength of messaging lies in its robustness, particularly over wide-area networks.

5 Assessment of individual advantages (Trees)

In this section we examine various individual claims related to mobile agents and consider arguments for and against their use. As a general statement, we have not discovered any client-server functions which are important for network services and which are uniquely enabled by the use of mobile agents. For almost every agent-based function proposed, we can propose an alternative based on existing protocols; this will become apparent in the discussions below. We believe therefore that the individual advantages of mobile agents are relative rather than absolute and the goal of our analysis is to determine whether these relative advantages are individually or cumulatively sufficient to warrant employing mobile agents as the basis for client-server applications and services.

5.1 Agents can provide better support for mobile clients

Mobile devices such as laptop and notebook computers, as well as emerging classes such personal communicators, have three characteristics relevant to this discussion:

- *They are only intermittently connected to a network, hence have only intermittent access to a server.* This is certainly true today, when most mobile access to networks is via circuit-switched lines, but may be less true in the future when wireless access to packet-switched networks will be prevalent. The advantage here lies in the mobile client's ability to develop an agent request - possibly while disconnected - launch the agent during a brief connection session, and then immediately disconnnect. The response, if any, is collected during a subsequent connection session.

- *Even when connected, they have only relatively low-bandwidth connections.* This is likely to remain true for some time to come. V.fast modems now provide 28.8 kbps links on dial-up lines, but wireless links to public networks are unlikely to exceed 12 kbps per client this century. The advantage here lies in the ability of an agent to perform both information retrieval and filtering at a server, and to return to the client only the relevant information. Thus the information transmitted over the network is minimized, which has strong cost implications for devices connected by public wireless networks.

- *They have limited storage and processing capacity.* While there are laptop computers today with 500 MB disks and Pentium-class engines, there will always be a class of devices that tries to make do with 'minimal' resources; for example, Hewlett Packard's successful HP95/100/200 series, which natively offers only 2 MB of storage. The advantage here lies in the ability of an agent to perform both information retrieval and filtering at a server, and to return to the client only the relevant information. Thus the information transmitted to the device is minimized and the device does not itself need to perform filtering.

There are thus two technical features of agents at play here:

- *Reduction of network traffic.* In the case of RPC-based communication, there are typically several flows between the client and server in order to perform even a simple transaction. In the case of secure RPC, there may be several tens of flows for a complex transaction. It is expected that these flows could be reduced to a single mobile agent with a corresponding reduction in network traffic, most importantly on the low-bandwidth access network.

- *Asynchronous interaction.* However, we have seen that this is a property of any message-based system, and does not require in itself programmable agents. Message buffering on the client device (inbound) and on the communication server (outbound) are well known features [3].

- *Remote searching and filtering.* If all information were stored in structured databases, it would suffice to send a message to the server containing SQL statements and perhaps perform backend filtering on the search results. Given that most of the world's data is in fact in flat, free text files, remote searching and filtering does require the ability to open files, read, filter and possibly develop an index. Agent programs are certainly a plausible method of performing this service. We wonder, however, if they are are the only way to perform this service. It would seem that a search engine installed at the server could achieve the same results, without requiring the dangerous generality of a programming language and execution environment.

Assessment: There is a real problem to be solved for mobile clients and mobile agents do have advantages for attaching mobile clients to networks. It is less clear that the entire network's servers need to be adapted to meet this need. Architecturally one would prefer to solve this problem at the edge of the network and make mobile clients as robust as non-mobile clients by providing proxy clients at the edge of the network [12].

5.2 Agents facilitate semantic information retrieval

We should look at remote searching and filtering further, since it is one of the central issues in agent programming. Consider a more sophisticated information retrieval system based on Semantic Retrieval. The user enters a query at his or her client device.

The system interprets this query semantically, possibly asking the user questions and getting clarification. This reformulated query is then transmitted via an agent to one or more servers, which retrieve information and present it to the agent, possibly getting additional feedback on the query and quality of the information retrieved.

In order to do this well, the system needs to be able to interact with both the user and the sources of information. Interaction with the user is easy, because he or she does not enter much information. Interaction with the multiple servers is not nearly so easy. First, the sources are likely to be distributed over many locations. Second, the amount of information involved (including most of the information which is in fact not relevant to the query, and hence should be filtered out) is huge.

The ideal system will possess knowledge specific to the domain in which it operates and specific to the user's interests, as well as the ability to filter data based on this knowledge. This knowledge will be based on exposure to a lot of current data in its area of expertise. It is far more efficient for the program extracting this knowledge to go to the source of the data instead of sending the data to the program, especially since the program is primarily filtering and summarizing the data. Thus an agent can do automatic indexing of documents, which will include identifying a small number of interesting documents from among a large number of uninteresting ones. It might also identify documents potentially interesting to other agents and inform them of this fact. Distributed indexes could be built up hierarchically, geographically, and by subject.

The difference between real semantic retrieval and simple keyword searches (which will differentiate offerings which people will pay for, from those currently available for free) is the amount of information which can be passed through the system to allow to it become a real expert in its field and to have high quality, current information at its disposal. Distributed intelligent agents, which are co-resident with the data sources and persist of their own accord (performing incremental indexing) have a real advantage in this respect over centralized, more static systems. To the extent that mobility allows the agents to get closer to the actual data source (e.g., real-time vehicular traffic data, weather data), mobility becomes a real advantage here too.

The counter argument to this is that it still appears possible to define standard retrieval and filtering programs, which could be installed at information repositories, and to send with the query a user context of previous searches that the user has found relevant. The information must be in any case be propagated among the various servers addressed by a query. This seems more efficient than propagating both the user context data and the search engine itself. It is true that it will be difficult to get agreement on a standard search engine, but then it will also be difficult to get agreement on a standard agent environment.

There is a second argument that in the future, bandwidth will be plentiful and cheap, and network carriers will be trying earn money by selling computation services. In this case performing the searching locally on the user's own client device may save money, provided the client is strong enough to perform this task. It can also be argued that mobile agents would allow the user to choose between free, local computation and for-fee vendor computation.

Assessment: This is an interesting new approach to information foraging in large networks. If mobile agent execution environments can be made prevalent throughout the network, this would offer a good support for this need. Equally however, the provision of intelligent search capabilities at all network servers using non-agent-based communication would seem to accomplish the same result.

5.3 Agents facilitate real-time interaction with server

Another reason for visiting a server is that it has an interface to a unique piece of external equipment, for example a machine tool. If the latency in network transmission is high compared to real-time constraints imposed by the external equipment, then it is desirable to send the controlling program to execute remotely on the server. An extreme case of this is the control software for space probes exploring the distant solar system. A program executing locally, even if interpreted, has a relatively low and certainly bounded latency and can provide more opportunities for error recovery.
Assessment: This seems very valid, although not in the mainstream of network services and hence not a major driver for this assessment.

5.4 Script languages provide better support for heterogeneous environme nts

Current networks are heterogeneous and will continue to be so. While developing, operating and maintaining heterogeneous network services is more difficult than the homogeneous case, it is less difficult than achieving or maintaining homogeneity in real-world environments. Passing data and commands among heterogeneous computers is more complex, but working solutions and exist and the actual number of useful permutations is not very high; heterogeneity among hardware and software in the same family is roughly as big a problem.
The use of a script language for program and data exchange enables the program and data representation to be independent of the platform, once the script environment has been ported to all necessary platforms. While this is a useful characteristic of script-based programming, it has little to do with mobile agents *per se*. The same advantages can be achieved by text representations of data or queries.
Assessment: This seems a weak argument.

5.5 Agent-based queries/transactions can be more robust

In current implementations, RPC communication is relatively fragile. RPC client-server computing was developed for LAN-based systems, where the application developers could make strong assumptions about the integrity of the LAN communication and of the availability of the server. Experience shows that when these client-server applications are extended over wide-area networks, they become less reliable. It seems likely that this problem was responsible in part for GMI's introduction of mobile agents [24]. Mobile agents offer two areas of advantage here:

- The messaging aspect, which provides reliable transport between client and server, without requiring reliable communication. While in principle unreliable communication layers can support RPC, the synchronous nature of this method means that re-transmission delays eventually become unacceptable.

- The recovery aspect, in which the dispatched agent is capable of dealing with the required server being unavailable, or unable to provide the required service. This supposes that the mobile agent program carries with it or knows how to access knowledge about alternate sources. To avoid this information becoming stale with time, it should either be provided to the agent by the dispatching client, which places a burden on the client to maintain this knowledge or otherwise access such a knowledge base itself, or the agent should itself know where to go and look for the knowledge.

  While this appears plausible for simple cases, say duplicate servers, it seems to encumber the mobile agent unless the recovery mechanisms can be made sufficiently general that they are supported by the base classes of the agent framework. In less simple cases, the alternate server may have more or less the same information or transaction service, but require the request to be expressed in a different form. Techniques to permit this kind of transformation are being developed using the Knowledge and Query Manipulation Language (KQML) [11].

  If this aspect of robustness is important, and it probably is, then RPC-based client (or server) applications can be made more robust in exactly the same way. If a given RPC request fails, the client can invoke exactly the same knowledge base to look for alternates and has the advantage that it can verify the alteration with the user,

Assessment: While this is a strong claim, it can equally be viewed as a motivation to make alternate communication protocols more robust for WAN-based network services.

5.6 Agent-based transactions & queries can be expressed more flexibly

One of the implicit hopes for intelligent agents is that they will enable (non-specialist) users to enter queries or transactions in natural language without knowing how or where the request can be satisfied. The agents will reformulate the concepts of the query into more precise terms and will identify one or more servers likely to be able to satisfy the request. A mobile agent will then be dispatched with the query and will presumably at some future time return the result to the delighted user.
This again has relatively little to do with mobile agents. The natural language support could be provided for any user interface. The matching of request to server could also be used to set up an RPC-based query. The transport of the request by the mobile agent has nothing to do with enabling the query to be expressed more flexibly.
Assessment: This has nothing to do with mobile agents *per se*.

5.7 Agent-based transactions avoid the need to preserve process state

The need to preserve the entire process state at a client and a server during each flow of a complex operation adds considerably to the burden of client-server computing. Unless the client and server applications provide methods for making the RPC state

persistent, which adds a significant performance overhead, failure of either client or server results in the loss of at least one and possibly many operations and recovery mechanisms at either end may generate useless network traffic. The ability of the mobile agent to carry its state around with it appears to relieve the sending computer of the need to preserve state. The state data are also simplified by performing the operation in an interpreted language rather than machine code.
However, the state must now be carried around the network with the mobile agent. This may on the one hand enrich the agent's interactions with servers, since it can express much more of the user's context, it can also result in the needless transmission of data that the agent will never use. This requires considerable skill on the part of the agent program developer to ensure that the mobile agent carries with it only the necessary and sufficient information.
Assuming that in the future this agent will return to the sending application (or one of the servers that it has visited will send a reply), the sending client must be able to relate the returning agent or other response to the original request. In other words, the sending application must preserve at least the application protocol state. This would seem to be roughly the same as the state information transmitted with the agent, modulo some fine details of the (interpreter) process state, so the saving may not be very great in terms of quantity of storage. However the mobile agent-based operation may be more recoverable than an RPC-based operation, because the operation is asynchronous, hence it can be re-started without strong time constraints and the saved state is at the application protocol, rather than the process level. The client application should, for example be able to re-create a 'lost' agent, which rises the question of how to detect that an agent has been lost. The usual sorts of protocols for distributed transaction processing seem relevant [20].
Assessment: There is an advantage here in the use of mobile agents in terms of the robustness of client-server operations, but it raises also an overhead question and also challenges for the efficient design of mobile agents *per se*. It is likely that similar robustness could also be achieved with alternative client-server methods.

5.8 Agents enable electronic commerce

Mobile agents here offer a number of useful possibilities:

- The agent can express the application-level protocol required to perform a transaction. This includes dialogues on choices and options, configurations, availability, delivery methods, and opportunities for selling up as well as the complete and accurate capture of the information required by the vendor in a particular format. Mobile agents are a plausible method for vendors to distribute the client end of a transaction protocol in a device independent way.

- Alternatively, the mobile agent may be able to present the consumer's desire as a query to a number of potential vendors to determine degree of match, price, availability, and so forth.

- The agent may also be able to consult a 'consumer guide' or other advisor before making a purchase.

- The agent can provide a secure vehicle for the transaction, providing bilateral authentication and privacy.
- The agent can provide a transaction currency for settlement; the agent's account is presumably reconciled periodically against 'real' money.

Of these, the last two are certainly readily replaceable by secure RPC or secure message-based client-server interactions. The first offers a much better facility for software distribution than exists today, although again its functions can be equally well performed by RPC or messaging. Today's methods for distributing application protocols include:

- Application clients preloaded by PC manufacturers.
- Application clients distributed in conjunction with other products, for example modems.
- Application clients downloaded from network sources such as CompuServe (tm.).

The ability of the vendor to distribute clients via an network, preferably to targeted consumers, offers significant reductions in the cost of capturing new customers (of the order of $50 per new customer for application preload). It also exposes the 'dark side' of mobile agents: junk clients, virus clients, dishonest clients.
The second provides a new opportunity, not readily implementable by conventional methods, which is to the advantage of the consumer, in that he or she is not locked into dealing with a particular vendor. In today's model, vendors wishing to support electronic transactions have been forced to integrate their servers with monolithic network services such as Prodigy or America On Line. The user can access only those vendors supported by the service. The vendor can reach only those users subscribed to the service. This 'closed' market place is giving way quite rapidly to an 'open' market place in which users and vendors engage in direct (spontaneous) commerce. Obstacles to the more rapid evolution of this open market place are:

- The difficulties of finding the vendors (lack of a good, global service directory).
- Lack of a common application transaction protocol or the ability of the user to easily acquire the vendor's proprietary application protocol (see obstacle no. 1).
- Lack of privacy and security, although vendors appear willing to perform experiments without solving these problems.

We may expect see numerous experiments in this area during 1994-95, particularly in the area of extensions to World-Wide Web servers [14].
The third possibility further extends the world of electronic commerce and its analogy with the 'real' world of commerce. We may anticipate a wide range of secondary commercial or quasi-legal services in support of electronic consumerism. As with the second possibility, the degree to which these can be established will depend on the degree to which the service providers wish to establish 'free markets'.
Assessment: Although mobile agents do not offer any technical advantage here, they do offer interesting convenience for vendors and service providers wishing to enable spontaneous electronic commerce, and could offer advantages to consumers.

5.9 Agent-based transactions scale better than RPC-based transactions

This is basically the RPC versus messaging argument described above. The asynchronous nature of mobile agents appears likely to enable higher transaction rates between servers, but a similar result could be achieved by messaging alone. On the other hand, the need to execute the agents and to support rigorous security around the agent execution environment could become significant computational loads in themselves. How many resident agents would, say, Dow Jones wish to support on its stock price servers? Is it really plausible that hundreds of thousands of agents sit there monitoring the ticker feed? Dow Jones may wish to sell the computational capacity to support this load, or alternatively, third-party servers, which receive the ticket feed from Dow Jones, may offer this as a valued-added service. Of course if Dow Jones can charge for the service of hosting a resident agent, this may be an interesting service business in itself.

Assessments: As a method of supporting simple queries and transactions, mobile agents benefit from the scalability inherent in messaging. Whether agent-based computing itself is efficiently scalable will depend on the extent to which service providers permit generalized computing by resident agents.

5.10 Secure agent-based transactions have lower overhead than secure RPC

The argument here is rooted in the fact that in general several RPCs are required to execute a given transaction, whereas the same transaction could be accomplished by a single mobile agent (presumably of roughly the same size as the total RPC traffic). The overhead of securing a single RPC is presumably similar to the overhead of a single secure agent, so the agent would appear to offer a technical advantage. In practice, secure RPCs are not used for every step of a secure transaction (because of the overhead), unless privacy is the main concern. If authentication is the main concern, practical RPC-based transactions will use a secure RPC only for the final commitment.

Whether the secure agent is more efficient than the secure RPC will also depend on the nature of the transaction; the use of agents may offer better scalability, but introduces much higher latencies.

Assessment: This does not seem very plausible.

5.11 Agents enable users to personalize server behavior

The view here is that servers should offer basic APIs and export them via the bindings of the agent execution environment for exploitation by mobile agent programs. The user (or other agent author) then has the freedom to use the server as he or she sees fit. Thus in the electronic commerce example, if the user wishes to browse the catalogues of several vendors rather than simply using the client application provided by the vendor, he or she has the freedom to dispatch a mobile agent to forage the vendors' servers for information relevant to a purchase. The dark side of this capability is the equal facility with which viruses and so forth could be introduced to servers.

Alternatively, the client could itself periodically execute a (script) program that would fetch information from servers using RPCs or messaging and perform the analysis locally. The arguments against this approach have been reviewed above; they relate to the efficiency of remote versus local filtering and the special problems of mobile computers.
Assessment: On technical grounds, setting the virus problem aside, this is a very valuable new capability. We wonder, however, whether service providers will really encourage this form of interaction, which dramatically reduces the control they have over their customers and reduces the server interface to that of information transactor. The virus problem remains significant.

5.12 Agents enable semantic-routing

Today's client-server interactions require detailed knowledge at both client and server of each other's application functions and communications protocols and addresses. One of the expectations expressed for mobile agents is to relieve the client and the user of much of these burdens. A user requiring specific information or any other service would express his or her needs in (something like) natural language and the query would be transmitted to a consultant agent. The consultant agent would reformulate the natural language query into the vocabulary and syntax of the Agent Communication Language. It would then consult its own index and possibly the indices of other consultants to identify one or more servers likely to be able to satisfy the query. The consultant would forward the query to these servers and the results would be returned directly to the requesting client. Alternatively, the results might be returned to the consultant which might then engage in a dialogue with the user to refine the search results. Thus the initial query submitted by the user is routed based on its semantic content. This example relates to information retrieval, but the same methods can be used for handling mail, transactions or indeed any of the documents handled by workflow systems.
Although mobile agents certainly facilitate several aspects of this process, there is again nothing here that can be performed exclusively by agents or indeed significantly better than by other means. The query routing process is in fact very similar to the AnyWhere proposal for semantic routing of messages [13]. The query reformulation process is purely a natural language activity that could equally well be applied to messages and could in principle be performed directly at the client rather than by a separate consultant. The indexing of server content is certainly facilitated by mobile agents, but can be accomplished in other ways. The submission of a reformulated query to multiple servers is well known in the absence of agents. The refinement of query results is certainly a good application for intelligent agents, but again is not specific to mobile agents.
Assessment: Although mobile agents offer no exclusive advantage for this application, nonetheless it is a very plausible application for mobile agents, because the same mechanism can be used to integrate clients, consultants and servers. One begins to see here the flexibility gained from this approach.

5.13 Mobile agents enable intelligent mail handling

Intelligent mail handling is the capability of the method and timing of mail delivery being determined by the semantic content of the mail item under the control of rules established by the recipient of the mail [12]. This is effectively a form of semantic routing. In the AT&T Personalink service [4]. is transported by courier agents, which are programmed in Telescript. Since the courier agent object class will be present at every agent execution environment, the courier agent actually needs only to identify itself as an instance of this class; there is no need to actually transport any method code in this case. The Personalink (or MagicMail) framework provides agent execution environments which can be visited by the courier agent and where the courier agent can engage in a dialogue with the host which routes the agent to the recipient's outpost (mailbox) in the current domain or directly to the recipient's domain (client device), if this is connected. The recipient's outpost may contain an intelligent agent which extracts key attributes from the courier agent and uses its rule set to determine how the courier item should be handled. The value of using a courier agent for mail transport rather than a simple mail transport protocol could lie in the general value of object encapsulation, in security services (authentication, non-repudiation, privacy, anonymity, payment), or in the ability to take part in translation services, although as usual, it is by no means clear that courier agents offer unique advantages for these purposes.

As with the semantic routing discussion above, there is nothing in this function which is intrinsic to mobile agents. The courier agent serves as the transport mechanism; the intelligent handling comes from the actions resulting at the intelligent agent from the mail arrival event. In ICS [12] the recipient is in control; the rules will be set up to reflect the recipient's handling preferences for certain events. In principle though, the recipient could enter a rule which says in effect: *Respect the sender's preferences.* The ICS Alter Ego could then examine attributes such as the priority assigned to the message by the sender, whether the sender requested that the recipient be notified (by paging) of the arrival of the mail, and so forth.

Assessment: Intelligent mail services depend on the processing of mail attributes by an intelligent agent. Mobile agents are a convenient transport mechanism for mail, but have no essential role in the attribute processing.

5.14 Agents can be prototypes for RPC applications

As was mentioned above, most applications which can be realized via mobile agents can also, often better, be realized via RPC. In order for this to happen, however, the application actually has to be implemented, standardized, and widely installed. This is, at best, a difficult and lengthy process. At worst, the application may be implemented poorly as standards constrain it before real experience with the application is obtained. In some cases, the application may never get off the ground at all, because it cannot gain wide enough acceptance without people actually seeing it work.

An agent-based interim implementation, on the other hand, can be done without a lengthy standards process. The agent is self contained and flexible. It is thus capable of functioning with relatively little coordination with existing software. Even though it may function less efficiently than an RPC implementation which is more tightly coupled to the resident software and even though it may be functionally constrained to preserve the security of the host system, it can still be used as the basis for a prototype implementation of the application which can be used as a proof of concept and a vehicle to evaluate features and tradeoffs in the application. On the basis of experience gained with this prototype, a more informed decision can be made as to whether it ultimately is best to embark on a standardized RPC-based implementation, retain the agent-based implementation, or abandon the application entirely.

Assessment Agent-based implementations offer the opportunity to rapidly prototype and refine an application more quickly and inexpensively than via RPC. The constraints which need to be imposed on the agents for security reasons and the inefficiencies imposed by relatively loose coupling with resident applications may lead to a pessimistic evaluation. Overall, however, it may be better to evaluate an application this way rather than simply by architectural discussions.

6 Assessment of Aggregate Advantages (Forest)

We have seen above that while there are many individual areas where mobile agents offer advantages, there are few if any overwhelming advantages among these and that in almost every case, an equivalent solution can be found that does not require mobile agents. However, if we stand back and look at the sum of these advantages, that is all the functions that a mobile agent framework enables, then a much stronger case emerges. Benjamin Grosof has referred to this as the "software engineering" argument and is essentially the point that *whereas each individual case can be addressed in some (ad hoc) manner without mobile agents, a mobile agent framework addresses all of them at once.*

Many of the counter arguments advanced above are of the form: "instead of using agents to do the work remotely at the server, you could just as well do it at the client". It may not matter in theory how we split up function between the client and the server, but it may be critical in practice, because clients and servers are controlled by different people, and work under different sets of constraints.

Consider PostScript (tm.), for instance (as GMI clearly did): it involves having a standard interpreter that runs on print servers. A user who wants something printed sends a program to the server, which executes it and produces the output. How valid would it be to argue that this is not really necessary, and the print servers themselves could be in charge of accepting and formatting passive input text, since if someone comes up with a new format for a document, one could just update all the printers in the world to know about the new format? It is true in theory, but absurd in practice. The statement "It is true that it will be difficult to get agreement on a standard search engine, but then it will also be difficult to get agreement on a standard agent environment" misses this point in a similar way: we only have to get agreement on a standard agent environment once, and after that everyone can write whatever clever search and foraging agents they want to. If the function is implemented in the server code, on the other hand, any new kind of operation (a cleverer search, a personalized foraging style, etc.) will have to wait for server updates before it can be used.

We have seen in the last 2 years what can be achieved by providing and disseminating a standard information server (WWW server) and clients. This has been so successful that Netscape, Inc. has recently announced that it will make Web browser clients available free (instead of at a proposed price of $99) and will charge only for the server

software. The HTTP protocol has been so successful that further progress in making information available via the Internet is only discussed in terms of extensions to Web servers and clients. Indeed it seems more than likely that experiments to provide mobile agent extensions to Web servers will be undertaken during 1994/95. Eventually one of these will be successful and will be deployed very quickly on thousands of servers. It would be no bad thing for IBM to go after this opportunity.
The argument is thus: "once we have reached agreement on how to provide a generalized, machine-independent execution environment which can bind to and enable the secure exploitation of server-specific capabilities, we will have created a completely general framework for network-based services, including:

- Information foraging,
- Semantic routing,
- Electronic commerce,
- Targeted dissemination of information, and
- Dissemination of the client side of application protocols."

In short, the framework is almost arbitrarily extensible to support network-based services.
The trick would appear to lie in:

- Doing the job well, and
- Getting it widely adopted.

GMI appears (so far) to have succeeded at the former, but may fail at the latter, because of the special role that AT&T plays in operating Telescript network services.
In summary, the lack of overwhelming strengths among the individual trees should not blind us to the overwhelming value of the forest as a whole.

7 Conclusions & Recommendations

- 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 when the network latency prevents real-time constraints being met by remote command sequences.
- The individual advantages of mobile agents therefore rest on relative technical and commercial factors compared to alternative methods. The technical advantages of mobile agents identified in this assessment are:
  - High bandwidth remote interaction
  - Support for disconnected operation
  - Support for weak clients
  - Ease of distributing individual service clients

  - Semantic routing
  - Scalability
  - Lower overhead for secure transactions
  - Robust remote interaction

- While none of the individual advantages of mobile agents given above is overwhelmingly strong, we believe that the aggregate advantage of mobile agents is overwhelmingly strong, because:

  - They can provide a pervasive, open, generalized framework for the development and personalization of network services.
  - While alternatives to mobile agents can be advanced for each of the individual advantages, there is no single alternative to all of the functionality supported by a mobile agent framework.
  - In addition to providing an efficient support for existing services, a mobile agent framework also enables new, derivative network services and hence new businesses.
  - Mobile agents are expected to appeal strongly to the Internet community, since they can provide an effective means for dealing with the problems of finding services and information and since they empower the individual user.

- The individual technical disadvantages of mobile agents identified in this assessment are:

  - Need for highly secure agent execution environments.
  - Performance and functional limitations resulting from security.
  - Virus scanning and epidemic control mechanisms.
  - Transmission efficiency, for example a courier agent compared to a simple SMTP mail object.

  The security and virus problems in particular require very close study and considerable technical innovation.

- Commercial issues raised by mobile agents include:

  - Difficulty of propagating agent execution environments onto large numbers of third-party servers.
  - Balance to be struck between open and closed electronic commerce.
  - Trust on the part of third-party server providers in the face of security concerns.
  - Willingness of the third-party server providers to permit users the ability to customize server behavior.
  - Willingness of the third-party server providers to support the computational load of mobile agents.

  - Perceived value among users.
  - Enthusiasm for this approach among the Internet community.

- We propose as the initial target for mobile agents a set of extensions to the World Wide Web server. This mobile agent environment should be integrated with an IBM WWW server and distributed free of charge to the Internet community. This free version would have limited functionality:

  - No security functions and no method of secure collection of fees.
  - Limited base classes (hence a performance limitation).
  - Limited number of platforms supported.

  This version would serve to demonstrate the capabilities, teach us the pitfalls, and establish the IBM scripting (and knowledge representation languages) as network standards. IBM could then exploit this beachhead to develop commercial versions which could be exploited by the same client population.

  There are many other possible targets for an initial development effort, but this appears to have the strongest impact. This is also a target where we expect many competing solutions.

- This assessment suggests further studies:

  - What degree of expressiveness can be safely accepted in an agent scripting language? Is it possible to devise languages that permit the expression of useful, quasi-general procedures, but which permit the non-existence of viruses to be proven?
  - How strong are the qualitative arguments for performance advantages? We could compare existing services with hypothetical mobile agent-based services?
  - Alternatively, what could be done to enable RPC-based client-server interactions to match the advantages of mobile agents.

The mobile agent approach continues to intrigue and shows signs of offering important qualitative advantages for network services. Assuming that solutions to the security problems can be found - and efforts are underway - the signs are sufficiently positive that we cannot rule out the possibility that mobile agents will be a successful new method of client-server interaction in network services. We are now engaged in developing plans to prudently explore this opportunity.

8 Acknowledgements

The need for this assessment emerged from many discussions of various topics related to intelligent and mobile agents. Many ideas expressed herein originated among members of the following group: Stephen Brady, Benjamin Grosof, Jeff Kephart, David Levine, the OREXX team, Colin Parris, Abhay Parekh, Phil Rosenfeld, Ted Selker, Steve White, Robin Williamson, and, of course, the Magicians.

References

[1] Selected Examples of distributed Applications. Number GG24-4167-00. IBM Corporation, Armonk, NY, 1994.

[2] No author given. *VIM Functional Specification Version 1.0.* Lotus Development Corporation, 1992.

[3] No author given. Oracle in Motion. Oracle Corporation, October 1994.

[4] Paula Bernier. Telescript's agents do the job. *Telephony,* 226(3):16, January 1994.

[5] A. Birrell and B. J. Nelson. Implementing Remote Procedure Calls. *ACM Transactions on Computer Systems* 2:39-59, February 1984.

[6] J. K. Boggs. IBM Remote Job Entry Facility: Generalized Subsystem Remote Job Entry Facility. *IBM Technical Disclosure Bulletin,* 752, August 1973.

[7] F. Cohen. Computer Viruses: Theory and Experiment. *Computers and Security* 6:22-35, 1987.

[8] R. J. Cypser. Communications for cooperating systems, pages 232-241. Addison Wesley, 1991.

[9] R. J. Cypser. Communications for cooperating systems, pages 244-245. Addison Wesley, 1991.

[10] M. Crowley-Milling et al. The Nodal System for the SPS. CERN, 78-87, 1978.

[11] T. Finin, R. Fritzson, D. McKay, and R. McEntire. KQML as an Agent Communication Language. In *The Proceedings of the Third International Conference on Information and Knowledge Management (CIKM '94)*, ACM Press, November 1994.

[12] C. G. Harrison. Smart Networks and Intelligent Agents. *Proceedings of MediaCom'95*, April 1995.

[13] Barron Housel. Anywhere. IBM NSD/RTP, 1993.

[14] T. Berners Lee, R. Cailliau, A. Luotonen, H. Frystyk Nielsen, and A. Secret. The World Wide Web. *Communications of the ACM,* 37(8):76-82, August 1994.

[15] Patty Maes and R. Brooks, editors. *A Biologically Inspired Immune System for Computers.* MIT Press, 1994.

[16] Gerald Neufeld and Son Vuong. Overview of ASN.1. *Computer Networks and ISDN Systems* 23(5):393-415, February 1992.

[17] J. K. Ousterhout. *TcL and the Tk toolkit.* Addison-Wesley Publication Company, 1994.

[18] J. Postel. Simple Mail Transfer Protocol, Request for Comments 821. August 1982.

[19] D. L. Presotto and D. M. Ritchie. Interprocessor Communication in the Eighth Edition UNIX System. Proceedings of the 1992 USENIX conference, June 1985.

[20] M. Sherman. Architecture of the Encina distributed transaction processing family. *ACM SIGMOD, International Conference on Management of Data*, May 1993.

[21] A. Tannenbaum. *Computer Networks, 2nd ed.* Prentice-Hall Publishing, 1988.

[22] Chii Ren Tsai and V. D. Gligor. Distributed Systems and Security Management with Centralized Control. *Proceedings of the Spring 1992 EurOpen/USENIX Workshop*, April 1992.

[23] J. E. White. Telescript Technology: The Foundation for the Electronic Marketplace. General Magic Inc., Mountain View, CA, 1994.

[24] James White. *RPC over WANs.* General Magic, Inc., private communication, August 1992.