An Architecture for Smart Internet Agents

Niranjan Suri, Kenneth M. Ford, and Alberto J. Cañas

Institute for Human & Machine Cognition
University of West Florida
11000 University Parkway
Pensacola, FL 32514
nsuri@ai.uwf.edu

Abstract

Considerable work has been done in the area of agents and agent technology in the past few years. The agent metaphor has been applied, with varying degrees of success, to a variety of situations from personal assistants to distributed problem solving. This paper describes a system designed to support agents on the Internet and addresses some basic issues dealing with network agents: Agent transport, agent security, and agent locator services. While work has already been done on these three issues, in most cases the existing systems and solutions do not address or scale well to the requirements of a large-scale, *uncontrolled* network such as the Internet. This paper explores transport and security issues for agents on the Internet as well as an innovative approach that uses the World-Wide-Web as a distributed knowledge base to help agents find other agents.

Overview

Agent technology and agent-based systems have been the focus of significant research and development efforts over the last few years. The definitions of what constitutes an agent vary widely and consequently so do systems that claim to be agent-based. This paper takes a network-centric view of agents, particularly mobile agents, which have the following characteristics:

- Independent, long-term existence (with agent state being preserved across multiple activations)
- Autonomous behavior, which allows agents to move around the network as they decide
- Mobility over a network (with the state of the agent being preserved across multiple transportations)
- Ability to replicate
- Adaptability (learning from previous experience)
- Represent and act on the interests of a user

The above characteristics separate traditional client-server or applet-based systems from agent-based systems. Mobile agents are a promising technology for many new kinds of applications ranging from exciting cyber-malls to the more mundane tasks of email filtering and network monitoring. While any of these applications could be implemented without agents, agents provide the most straightforward approach. They also provide other distinct advantages such as reduced network bandwidth, better support for mobile clients, real-time interaction with servers, and so forth. In general, agent-based systems can replace several of the existing mechanisms for distributed processing and information exchange as well as provide metaphors for new kinds of applications.

Requirements for Agent-based Systems

Agent-based systems have to satisfy a number of requirements before they can be used extensively on the Internet. Extensive use implies possibly many agent writers, many agents, and many agent execution environments, all interacting with each other. One of the underlying problems in such a network is that most of the time the different parties (writers, agents, or hosts) do not know or trust each other.

As a motivating example, consider the requirements to replace email with agent-based communication. Suppose a person wanted to send a message (to provide some information, fix a meeting time, ask for a copy of a publication, etc.) to another person. Instead of typing an email message, the person would write an agent (or pick from a library of available agents) and send the agent to the recipient. The agent would then interact with the recipient or other agents that represent the recipient (such as an appointment manager agent or publications archiving agent). Therefore, instead of having an inbox that contains a list of messages, users would have an inbox of agents waiting to communicate. Such a system would imply many users, many agent-writers, many different platforms, and many execution environments all wishing to interoperate. This is only one example of how agents could be used in a widespread manner on the Internet.

In order to support such systems, agent-based systems would have to satisfy the following requirements:

- A popular, easy to use language for writing agents
- Cross-platform support for agents
- Support for agents that have not been certified or identified and authenticated
- Secure execution environments for untrusted agent code
- Support for agent mobility
- Mechanisms for inter-agent communication
- User interface mechanisms for communication between agents and users

- Agent locator services
- Access to Internet resources through standard Internet protocols and standards

Current Systems and Their Limitations

Several agent systems have already been developed, some commercial and others free. One of the first systems to implement agents (as qualified by the definition above) was Telescript (White 1994) from General Magic. General Magic has since ported much of the Telescript functionality to a Java-based system called Odyssey (General Magic, Inc.). Other examples of commercial systems include Voyager from ObjectSpace (ObjectSpace, Inc.), Aglets from IBM (Lange and Chang 1996) (Lange 1997), and Concordia from Mitsubishi Electric ITA (1997a and 1997b). A more recent development is the Mobile Agent Facility specification that will become part of OMG's CORBA standard (GMD FOKUS and IBM Corp. 1997). There are freely available systems as well such as Agent-Tcl from Dartmouth College (Gray 1995), and Tacoma from Cornell University (Johansen, van Renesse, and Schneider 1995).

The major limitation of Odyssey, Voyager, Aglets, Concordia, and other systems is that they do not support secure execution of untrusted agent code. These systems are provided as toolkits to programmers who can construct agent-based systems to run in controlled environments. They are not designed to support a scenario where an untrusted user on the Internet is able to send an untrusted agent to execute on a remote platform. They would either require that users authenticate themselves (which is impractical given the number of users on the Internet) or that the agents are produced in a controlled environment where they can be certified or trusted.

Another problem with the above systems is that they do not address the problem of agents finding other agents. If agents are to be deployed in significant numbers (and at numerous locations) to act as producers and consumers of information, agents need to be able to find each other. If the Internet were to evolve to support agent-based computation but retain its current unstructured organization, people would begin to arbitrarily setup agent execution environments (just as people do now with web servers and web pages). In such an environment, agents would be faced with a "where to go" problem: Given that agents can move from system to system, how do agents find which systems to go to when looking for specific information or services? People often waste time "lost in cyberspace" not able to find the information they need. Agents would also face the same problem and waste significant network bandwidth and processing time by wandering from server to server.

Advantages of Proposed Agent Architecture

The architecture proposed in this paper is specifically tailored to scenarios that require widespread use of agents on the Internet. The agent server supports many of the requirements listed in the earlier section providing key features such as:

- Support for anonymous agents
- Extensive, configurable security for agent-execution environments
- An agent transport mechanism
- Agent-user interaction mechanism
- Agent locator service based on WWW hotlists

In addition, the free availability of source code (for non-commercial use) makes the system well suited for further research and development. The non-availability of source code for commercial products such as Odyssey, Voyager, and others makes it difficult for other interested parties to develop and evolve these products.

Architecture Details

Figure 1 shows the overall architecture for this system. The architecture is designed around the notion of agent server daemons that are located on various hosts on the Internet. These servers provide a secure environment for agents to exist and operate. Agents can move between the daemons and preserve their state. The agents normally originate from personal agent servers that are running on the desktops of user workstations. The following steps illustrate the typical life of an agent:

- A user starts running an agent in the user's personal agent server
- The agent can interact with the user to obtain initial parameters (this interaction relies on the capabilities of the personal agent server)
- The agent may then move from the personal agent server to agent server daemons as necessary to accomplish its task
- Once the agent has completed its task, the agent moves back to the user's personal agent server to report its results

The agent server daemon has been implemented as a UNIX

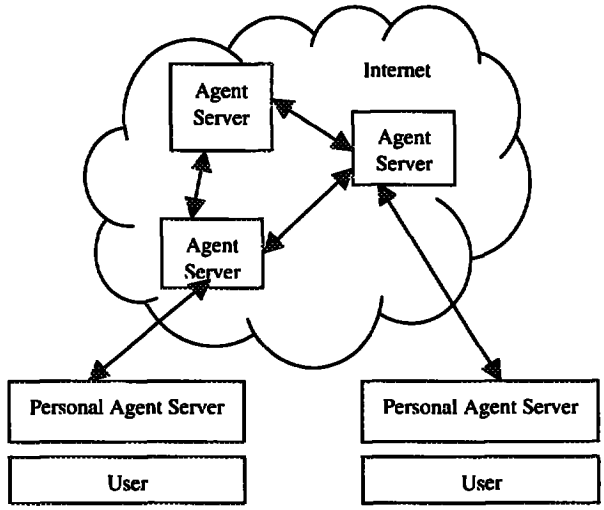

Figure 1 : Overall Architecture of the Agent Server System

daemon and has been ported to run on Solaris, SunOS, AIX, IRIX, and Linux. Ports to other UNIX platforms should be reasonably straightforward. Once started, the daemon runs continuously in the background waiting for connections. Other daemons or personal agent servers may connect to the daemon and transfer an agent for execution. Therefore, this daemon is executed on systems that wish to host foreign agents or long-lived local agents.

The personal agent server has been implemented as a Motif application running on UNIX systems. It is designed to allow easy porting to a number of other operating systems including Windows and Macintosh. Usually, each user starts up a copy of the personal agent server whenever the user wishes to interact with, send, or receive agents that were previously sent out to other servers. The personal agent server need not be continuously running. Therefore, agents wishing to return to the personal agent server from other servers may be queued up until the personal agent server is available. When agents return to the personal agent server, they are placed into a queue of incoming agents (similar to an incoming mail queue). Users may then select among the waiting agents and interact with them.

Anonymous Agent Services

In order to be practically usable on the Internet, the agent server daemon supports two different kinds of agents: user agents and anonymous agents. User agents are those sent by users who have accounts and privileges on the system hosting the agent server. These agents run using the same rights afforded to the user by the system. For example, the agent starts executing in the home directory of the user and is free to access or modify any files that belong to or are accessible by the user. The agent can also execute any programs on the system that would normally be available to the user.

If agent servers were to be widely deployed on the Internet, it would be impossible for all users who wanted to send agents to have accounts on the various systems. Therefore, the agent server also supports the notion of anonymous agents. Anonymous agents can be sent by anyone regardless of whether that person has an account on the system. The agent servers run anonymous agents in a restricted environment providing them with only limited access to the resources of the host computer. Extensive configurable security and quota mechanisms are available to allow administrators to setup and control anonymous agent services. Security mechanisms are discussed in a later section.

Agent Scripting Language

Figure 2 shows an agent written in the Tcl language. Tcl (Ousterhout 1990) was chosen as the agent scripting language for several reasons. Firstly, Tcl was designed to be embedded inside other applications, which was perfect for the needs of the agent server. The agent server instantiates a new Tcl interpreter whenever it needs to execute an agent's code. More importantly, Tcl allows the language definition to be changed by adding, replacing, and removing keywords without modifying the interpreter. Even all of the standard programming language constructs such as variable assignment, iteration, selection, and so on can be modified by the application embedding the Tcl interpreter. The agent server adds several keywords to the standard Tcl language such as *Go*, *Send*, *Find*, and *Window*. The agent server can also remove or replace certain commands to restrict the capabilities of agents. In addition, Tcl provides access to the internal data structures of the interpreter, which is necessary for the agent server to implement agent transport.

```
proc main {} {
    Go amaru remoteMain guest letmein
}

proc remoteMain {} {
    global list
    set list [exec /bin/who]
    Go tupac showlist guest letmein
}

proc showlist {} {
    global list
    puts $list
}
```

Figure 2: An agent that queries users logged into a remote system

Agent-User Interaction

The personal agent server runs inside a graphical user environment and provides an easy to use interface to the agent system. Users can start up the personal agent server in order to run agents or to send agents to other servers. An agent running inside this server can interact with the user by opening "conversation" windows. Agents can use the *Window* command to create new windows, send output to windows, and receive user input from windows. Figure 3 shows the agent version of the "hello,world" program. From the agent writer's point of view, interacting with the user is very simple and only involves reading and writing strings. Figure 4 shows the agent interaction window that was created by another agent.

Only agents running in personal agent servers are allowed to interact with users. If an agent is running on a server daemon, that agent has no mechanisms available to communicate with users. Agents are required to move back to the originating personal agent server and then present any information to the user.

Security Mechanisms

Since the agent servers have to guard against possibly

```
proc main {} {
    set wid [Window create]
    Window puts $wid "hello, world\n"
    Window puts $wid "\nWhat is your name?\n"
    set name [Window gets $wid]
    Window puts $wid "Hello to you too, " $name
}
```

Figure 3: An agent Version of the Traditional hello, world *Program*

hostile agents, the servers implement extensive security mechanisms. The security measures required can be divided into three categories: security for user agents, security for anonymous agents, and security for agents in the personal agent server. In addition, there are security issues dealing with the actual transport of agents.

The security mechanisms required for user agents are reasonably straightforward. Each agent is given all the access privileges of the user who owns the agent. Before a user agent is accepted by a server daemon, the user has to supply a password to authenticate the agent. Once the agent is authenticated, the agent runs as a process owned by the user who sent the agent. From that point onwards, UNIX will take care of enforcing access privileges for the agent.

In addition, the server administrator can enforce limits on the resources that may be consumed by user agents. These limits are implemented by a set of configurable parameters that are shown in figure 5.

The security requirements for anonymous agents are more critical. Administrators can apply different security policies for anonymous agents. In addition to all of the parameters listed in figure 5, the server allows the root directory for anonymous agents to be changed thereby allowing only a subset of the filesystem to be visible to agents. All anonymous agents also run in special accounts created on the servers. The agent server "sanitizes" accounts used by agents after the agents leave so that future agents do not accidentally "find" information left by previous agents.

Unlike the server daemon, which relies extensively on UNIX for most of its security mechanisms, the personal agent server was designed to be ported to a variety of operating systems such as Windows and Macintosh. Since most PC-based operating systems do not provide much in terms of security mechanisms, the personal agent server does not allow the execution of any untrusted agents. Only agents that were started by the current user are allowed to execute. Also, only agents that originated from a particular personal agent server are allowed to return to that personal agent server. The server detects tampering of agent code by using a CRC-32 checksum algorithm.

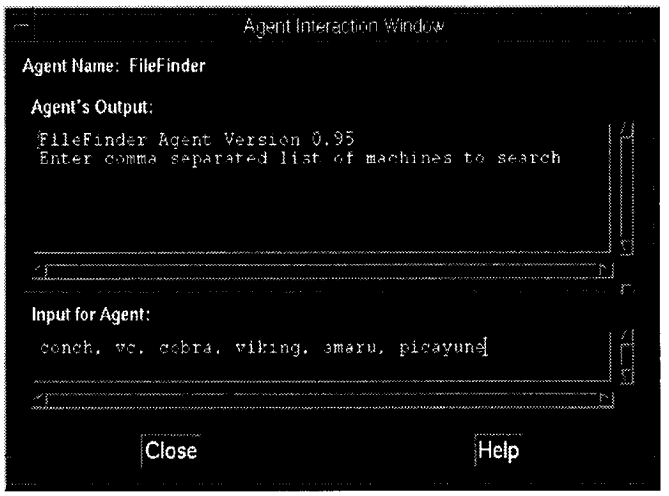

Figure 4 : An Agent Interaction Window in Personal Agent Server

Agent Locator Services

One of the main goals of the agent server architecture is to address the "where to go" problem often faced by people and programs navigating the Internet. The Internet as it exists today has no explicit structure or organization based on content. The only structure is that which is imposed by the Domain Name System (DNS) but this structure is based on physical location and organization rather than on content. For example, DNS allows us to find all computers in a domain such as uwf.edu but there is no way to find all computers that have information on Mars. Since any host on the Internet can act as a server and provide any kind of information, providing a content-based directory service is difficult at best.

Given the amount of time wasted by people looking for information, it would be exceedingly difficult for people to design useful autonomous agents that are supposed to work on the Internet. Agents will probably end up wasting time by "hopping" around the network from host to host. While this behavior might be reasonable on a small prototype network, it would be totally inadequate if agents are to ever come into extensive use on the Internet. One can imagine the total chaos caused by millions of agents blindly stumbling around from host to host looking for information. In fact, the benefit of reduced network bandwidth provided by mobile agents would be completely lost.

Recently, several hierarchical "hotlists" have emerged on the Internet (the prime example being Yahoo). These hotlists organize WWW servers based on their content into a large hierarchical tree. Such hierarchical organization provides contextual information that allows people to retrieve information in a more accurate manner than traditional search engines.

If, in the future, agent-based interfaces are setup as alternatives to information and services that are currently

MaxAgents:	The maximum number of concurrently running agents
NiceLevel:	The nice level to be applied to the agent process
MaxTimeToLive:	The maximum period of time that agents are allowed to live
AllowDuplication:	Whether agents are allowed to duplicate themselves
MaxDuplications:	Maximum number of duplications allowed for an agent
MaxLinesOfCode:	Maximum number of lines of code allowed for the agent's script
MaxNumberOfVars:	Maximum number of variables allowed for the agent
MaxVarNameSize:	Maximum size allowed for a variable name
MaxVarSize:	Maximum amount of storage that can be used by one variable
MaxTotalVarSize:	Maximum amount of storage that can be used by all the variables
MaxDataSize:	Maximum size of the data segment of the agent process
MaxStackSize:	Maximum size of the stack segment of the agent process

Figure 5 : Configurable Security Parameters for the Agent Servers

WWW-based, then the location of WWW-based information would coincide with the location of agent platforms that also provide information on the same topic. Therefore, if agents are able to locate sites that offer WWW-based information on a particular topic, then agents would also be able to find an agent platform at the same site that would provide the same service. A WWW bookstore that currently offers books on, say, music, would likely have an agent-server with software agents that sell music books.

The solution proposed in this paper is to provide a mechanism to the agents that would allow them to locate agent-based information or service providers by looking for corresponding WWW-based services. This scenario is different from systems that actually use the Web to find information. The information in the hotlists is not the ultimate data consumed by the agent, but is used as meta-information to locate agent-based services. Therefore, the hotlists and the Web acts as a distributed knowledge base and an external representation that tells agents where to go and look for information. Since the hotlists are an external representation that are updated independently of the agent system, agents can take advantage of the efforts of many people to organize and categorize the servers on the Internet into hierarchical lists. Any updates to the lists will automatically result in the agent system being updated. Also, as the structures of the lists improve, so will the performance of the agent system.

Another advantage of the hierarchical lists is that they have contextual information that can be provided to the agent along with the results of a search. For example, if an agent does a search for the string "mice," the server might return a set of possibilities falling in two contexts: Computer pointing devices and animals. The agent can use this contextual information to select among the possible matches.

The agent server system currently provides a stand-alone application that takes hierarchical lists and builds or updates a representation. The system does not try to merge information in several lists into one representation. Instead, an independent representation is built for each list configured for use in the server. Agents then query these representations (through Tcl commands provided by the agent server) as and when the information is needed.

Conclusions

The agent server architecture described in this paper addresses some of the basic requirements for implementing an agent-based prototype system on a large-scale network such as the Internet. In addition to addressing agent transport and security issues, the system provides an innovative approach to solve the "where to go" problem. A few simple agents have been written to test and exercise the system.

Work is currently underway to extend this project to allow Java to be used as the agent scripting language, add additional capabilities to agents, and provide better content-based directory services. Recently emerging standards such as Resource Description Framework (RDF) (Lassila and Schwik 1997) and Meta-Content Framework (MCF) (Guha and Bray 1997) are also being incorporated into the system and will possibly allow the agent locator services to work with much greater accuracy. Work also needs to be done to incorporate agent communication frameworks such as KQML (Finin et al. 1994), and KAoS (Bradshaw et al. 1997).

References

White, J.E. 1994. Telescript Technology: The Foundation for the Electronic Marketplace. *General Magic White Paper.*

General Magic, Inc. On-line reference at http://www.genmagic.com/agents/odyssey.html.

ObjectSpace, Inc. On-line reference at http://www.objectspace.com/voyager.

Lange, D.B., and Chang, D.T. 1996. *IBM Aglets Workbench : Programming Mobile Agents in Java.* On-line reference at http://www.trl.ibm.co.jp/aglets/whitepaper .htm.

Lange, D.B. 1997. IBM Aglet Application Programming Interface (J-AAPI) White Paper. On-line reference at http://www. trl.ibm.co.jp/aglets/JAAPI-whitepaper.htm.

Mitsubishi Electric ITA. 1997a. Concordia: An Infrastructure for Collaborating Mobile Agents. In Proceedings of the First International Workshop on Mobile Agents, Berlin, Germany.

Mitsubishi Electric ITA. 1997b. *Mobile Agent Computing: A White Paper.* On-line reference at http://www.metica.com/HSL/Projects/Concordia/MobileAgentsWhitePaper.pdf.

GMD FOKUS and IBM Corp. 1997. Mobile Agent System Interoperability Facilities Specification. OMG Technical Committee document orbos/97-10-05. On-line reference at ftp://ftp.omg.org/pub/docs/orbos/97-10-05.pdf.

Gray, R.S. 1995. Agent Tcl : A Transportable Agent System. *Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM 95),* Baltimore, MD.

Johansen D.; van Renesse, R.; and Schneider, F.B. 1995. An Introduction to the TACOMA Distributed System Version 1.0, Technical Report 95-23. Dept. of Computer Science, Univ. of Tromsø, Norway.

Ousterhout, J.K. 1990. Tcl: An Embeddable Command Language. *Usenix Conference Proceedings,* Winter 1990.

Lassila, O., and Swick, R.R. 1997. Resource Description Framework (RDF) Model and Syntax. On-line reference at: http://www.w3.org/TR/WD-rdf-syntax.

Guha, R.V., and Bray, T. Meta Content Framework Using XML. On-line reference at http://www.w3.org/TR/NOTE_MCF_XML.

Finin, T.; Fritzon, R.; McKay, D.; and McEntire, Robin. 1994. KQML as an Agent Communication Language. *Proceedings of the Third International Conference on Information and Knowledge Management.* ACM Press.

Bradshaw, J.M.; Dutfield, S.; Benoit, P; and Woolley, J.D. 1997. KAoS: Toward An Industrial-Strength Open Agent Architecture. *Software Agents.* Menlo Park, CA: MIT Press.