Transportable Agents Support Worldwide Applications

David Kotz Robert Gray Daniela Rus

Department of Computer Science
Dartmouth College
Hanover, NH 03755
{dfk,rgray,rus}@cs.dartmouth.edu

Abstract

Worldwide applications exist in an environment that is inherently distributed, dynamic, heterogeneous, insecure, unreliable, and unpredictable. In particular, the latency and bandwidth of network connections varies tremendously from place to place and time to time, particularly when considering wireless networks, mobile devices, and satellite connections. Applications in this environment must be able to adapt to different and changing conditions. We believe that transportable autonomous agents provide an excellent mechanism for the construction of such applications. We describe our prototype transportable-agent system and several applications.

1 Transportable autonomous agents

A *transportable autonomous agent* is a named program that can migrate from machine to machine in a heterogeneous network. The program chooses when and where to migrate. It can suspend its execution, move its code and state to another machine, and resume execution on the new machine. Transportable autonomous agents are well suited for the construction of worldwide applications for two reasons:

• **They are transportable.** In the traditional client-server paradigm, the client code communicates with the server code to access data and resources on the server's machine. For many applications, however, it makes more sense to move the code to the data. Moving the code to the data is particularly helpful if the client needs to sift through a lot of data or control a low-latency device, especially if the server's interface does not support the client's needs exactly. In information-retrieval applications, for example, the client's search function can be uploaded to the back-end database server. In user-interface applications, the application's interaction routines can be downloaded to the user's workstation. A transportable agent can jump to the remote machine and interact with the desired services much more efficiently than over the network.

Mobile code is especially useful when machines are poorly connected, such as in wireless networks, certain long-haul networks, and laptop computing. A transportable agent, for example, can jump off of a laptop and roam the Internet in service of its user, even if the user later powers down or disconnects her laptop. When the laptop reconnects (perhaps at a different location), the transportable agent jumps back with the results of its travels.

• **They are autonomous.** For a transportable agent to interact with other agents and systems, which are often far away from its owner and its "home" machine, it must be able to act autonomously. Indeed, its owner may have logged off, or the "home" machine may be temporarily disconnected or unreachable. Ideally, the agent should be able to autonomously and adaptively formulate plans for navigation, error handling, and

This research was supported by ONR contract number N00014-95-1-1204 and AFOSR contract number F49620-93-1-0266.

so on. In this respect we can borrow from the extensive experience of the artificial-intelligence and robotics communities. To be able to make plans and adapt to changing conditions, agents must be able to sense their environment, such as the current state of networks, the load on the computer, and the location of other agents.

2 Dartmouth's Agent Tcl project

At Dartmouth we are applying a unique combination of operating systems, artificial intelligence, and robotics experience to the design of a transportable autonomous agent system and several demonstration applications. The keystone of our work has been the development of *Agent Tcl* [Gra95, Gra96, GKCR96], an extension of the Tcl scripting language [Ous94]. An Agent Tcl program (agent) can simply "jump" from machine to machine, with the interpreter saving and transferring the state to the remote machine, which restarts the agent from that state in a new interpreter. We are currently extending the concept to a second language, Agent Java.

We have several auxiliary projects underway involving agent communications, navigation, disconnected operation, security, resource allocation, and graphical interfaces. We describe each briefly.

2.1 Communication

Agent Tcl provides the ability for agents to communicate through direct connections and through message passing. We have two higher-level agent communication mechanisms on top of these primitives.

Agent remote procedure call (ARPC) [NCK96] allows server agents (i.e., those which wish to advertise a service) to register with a "name server" agent by specifying their interface using a flexible definition language. Client agents specify the desired service by providing a similar specification. The name server matches the two specifications and provides a list of server agents that "speak" the interface. As a result, agents can find and communicate with other agents simply by agreeing on a common functional interface. By naming and typing all parameters and return values, allowing servers to specify default values for some parameters, and allowing partial matches, a single client can connect to servers implementing a wide variety of interfaces. We expect to define a few minimal interfaces that all agents support.

A **paging** mechanism allows one agent to locate and communicate with another agent *even if* the latter agent has moved around the network. Our current solution depends on the destination agent registering its position with its "home" machine after each jump. Other agents then find the agent by asking the home machine.

2.2 Navigation

Agents can jump around the network, but where do they go? How can they find places to visit and adapt to a changing network? We are focusing on three related services.

A hierarchical set of **navigation agents** maintain a database of service locations [Gil96]. Services register with these navigation agents. An agent looking for a service queries a navigation agent, which suggests a list of services (based on a keyword match) and possibly other navigation agents (that may be specialists in listing services on the requested topic). Later, the agent provides feedback about which services were useful, which the navigation agent uses to adjust its database to provide a better response to future queries.

A set of **network sensing tools** allow an agent to determine whether its host is connected and to estimate the latency or bandwidth to remote hosts. The simplest tool uses `ping` to the local IP broadcast address as a way of determining whether the local host is connected. The other tool, a *traffic monitor agent,* attempts to provide estimates of latency or bandwidth to remote hosts by tracking information about all recent communications (bytes moved, time required). Currently, our traffic monitor agent monitors only agent jumps, and uses a simple weighted heuristic to form an estimate. Observations are weighted according to their distance from the desired destination, both in time (more recent observations being given more weight) and in space (observations involving the destination machine being given more weight than observations involving another machine in the destination domain). We are still tuning our heuristic; we expect that our estimates will be rough but usable for large-grain decisions (e.g., using the estimates, and a greedy strategy, an agent would likely visit all sites in Europe before hopping to Australia to visit sites there [RGK96]).

2.3 Disconnected operation

Agents are ideal for partially disconnected environments, such as laptops, modem-connected home computers, and personal digital assistants, because they can travel the network while their "home" machine is disconnected or unreachable. For agents trying to jump into or out of the laptop, however, the traditional approach (try, timeout, sleep, retry, ...) can often fail, particularly if the agent does not happen to retry its jump during a brief reconnection period. To overcome these problems, our **laptop docking system** [GKN+96] pairs each laptop with a permanently connected *dock* machine (Figure 1). While not all machines act as docks, all machines have a *dock-master* agent.

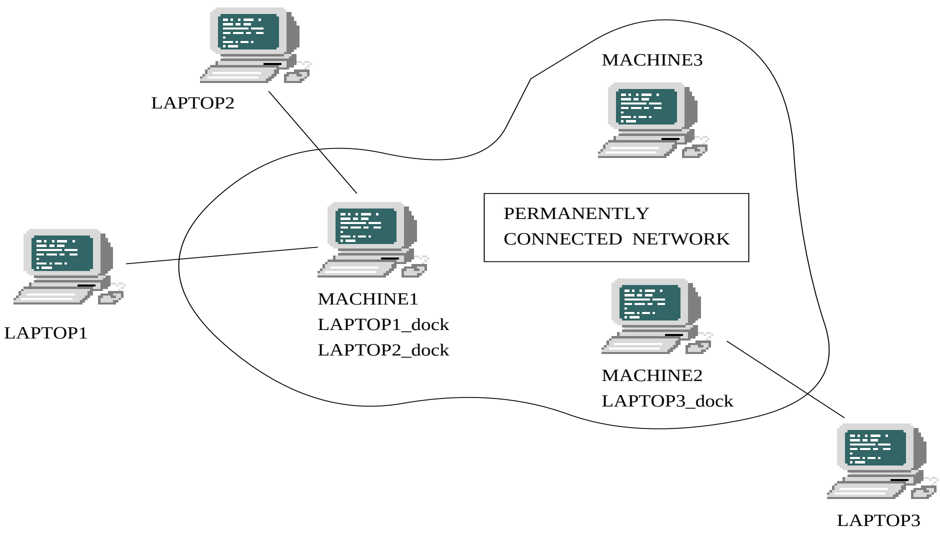

Figure 1: Laptop-docking system

Consider an agent wishing to jump to a disconnected laptop named D (Figure 2). To do so, it executes the command *agent_jump D*. When the command completes, the agent will be running on D; the process is transparent. The *agent_jump* implementation attempts to contact D, which fails because D is disconnected. So it then attempts to contact the dock-master agent on the laptop's dock. By convention, the dock for host D is named *D_dock*. Internet host naming allows a single permanently connected machine to have many aliases, which allows one host to act as a dock for many laptops. Once the agent is transferred to *D_dock*, it is not restored into a running agent, but stored on disk under the control of the dock-master at *D_dock*. When D reconnects, its dock-master agent contacts the dock-master at *D_dock* so that all waiting agents can be transferred to the laptop D, where they are restored. In the process, *D_dock* learns of any change in the address for D. Thus, agents trying to reach D will fail to reach it at its old address, jump to *D_dock*, and eventually reach D at the new address.

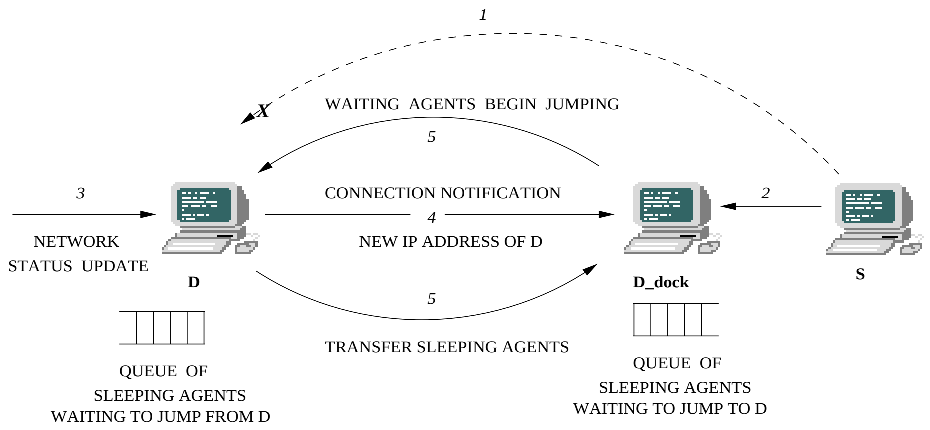

Figure 2: Jumping to or from the laptop

Now consider an agent trying to leave the disconnected laptop D. Again the agent executes *agent_jump*, which detects that the laptop is disconnected, saves the state of the agent to disk, and informs the local dock-master agent. The dock-master continually monitors the network status; when the network is connected, the dock-master immediately transfers all waiting agents off of the laptop (Figure 2). This scheme has several advantages: the agents leave the laptop as soon as possible; the agents do not miss any opportunities to leave; the agents are saved on disk, where they survive crashes and shutdowns, and do not occupy precious memory and CPU time; and their state is captured and ready for transfer as soon as the network is connected.

Thus, agents wishing to jump on or off the laptop move quickly as soon as the laptop is connected, minimizing the connection time necessary. Again, the entire process is transparent to the agent.

Note that an agent may jump from laptop to laptop, and both laptops need not be connected at the same time. In addition, if an Internet problem makes *D_dock* inaccessible when S happens to connect, the agent may jump from S to *S_dock*, then to *D_dock* when that becomes reachable, then finally to D when it reconnects.

2.4 Security

Agents can be **encrypted** and **digitally signed** using PGP. A host authenticates an incoming agent using the digital signature. **Resource manager agents** are long-lived agents that must be consulted by an agent desiring access to a resource. A resource manager implements the access policy for each critical resource, such as the file system, the screen, the speaker, and the keyboard [Gra96]. Agents must negotiate with the appropriate resource manager before they can access the resource; the resource managers determine the allowable access based on the agent's authentication. The Agent Tcl interpreter uses Safe Tcl [LO95] to ensure that an agent can neither bypass the negotiation step nor violate the restrictions provided by the resource managers. In effect, Safe Tcl divides the interpreter into two separate interpreters, a "user" interpreter and a "kernel" interpreter. The agent executes in the user interpreter; all negotiation and access routines are in the kernel interpreter and can only be called through an *exported* procedural interface. The decisions of the resource manager are cached within the kernel interpreter so that the agent does not have to negotiate for every access (e.g., for every read or write to a file). The strength of this security scheme is the clear separation between the policy providers (resource managers) and the policy enforcer (Safe Tcl). This separation allows the rapid introduction of new security polices and makes it much easier to integrate a new language. When Java is added to the system, for example, the standard Java security mechanisms will be used to enforce the *same* policy provided by the *same* resource manager.

In addition to the resource managers, each machine may have a *console* agent, which serves two purposes. First, it tracks all of the agents that are arriving or executing on the machine, and allows the user to terminate or deny entry to a particular agent. Second, it serves as a central point through which the resource managers ask the user if a particular access request should be allowed; eventually the user should be able to specify exactly those situations in which she should be asked.

The current security mechanisms successfully protect a machine from a malicious agent. Work is underway to protect an agent from a malicious machine (to the extent possible) and to protect groups of machines from a malicious agent. Our approach for protecting machine groups controls global resource allocation using electronic currency.

2.5 Resource allocation

In a dynamic and distributed system, it is often difficult to control resource usage, particularly when the processes are agents that migrate from host to host. In particular, we wish to prevent "runaway" agents that, intentionally or unintentionally, roam the network forever.

We propose to use a **currency model** of resource allocation in which agents "buy" needed resources using some universal currency units (which may or may not be tied to legal currency units). Needed resources include CPU time, memory, disk space, network bandwidth, screen space, database access, and information. When an agent is created, it is given a fixed amount of currency from its creator's own finite currency reserves. The agent then chooses how to best spend the currency to meet its goals. When it runs out of currency, it is sent back to the home site, which either provides more currency or terminates the agent.

Resource managers collect the currency spent on their resources. This accumulated currency may be spent by the owner of the resources, perhaps the person or organization owning that workstation, by giving the currency to its own agents.

Our hypothesis is that the effects of supply and demand will control prices, and that currency will allow the easy sharing of diverse resources without a complex policy controlled by a central machine or organization. We have embedded an auction-based negotiation model into a module that can be used by both client and server agents when negotiating prices.

Our currency is represented as cryptographically-protected digital cash, issued by one of a collection of banks. An agent trusts its bank, and banks trust each other. An agent normally has a "wallet" containing numerous cash "bills" of different sizes, which it can give to another agent with no bank involvement. The bank must be involved if a bill must be split into smaller bills, or if an agent wishes to verify that a bill is valid and has not already been spent.

2.6 Graphical interfaces

Agents can use Tcl's Tk toolkit [Ous94] to create graphical interfaces and interact with the local user. In addition, we plan to develop an **agent-composition tool** that allows programmers to build new agents by visually combining existing agent components. The tool would combine the code for the existing components with the appropriate glue code. This idea is similar to popular visual-programming languages.

3 Related work

Our work builds on previous work in agents [Age94], primarily in the AI community, and in transportable code, primarily in the systems community.

Mobile agents can be viewed as an extension of the remote procedure call and remote programming paradigms. Remote procedure call (RPC) allows a client to invoke a server operation *using the standard procedure call mechanism* [BN84]. Remote programming allows a client to send a *subprogram* to a server. The subprogram executes on the server and sends its result back to the client. Variants of remote programming include the Network Command Language (NCL) [Fal87], Remote Evaluation (REV) [SG90], and SUPRA-RPC [Sto94]. Agents generalize remote programming to allow arbitrary code movement. Our system allows agent programmers to choose between a remote conversation (with RPC, message-passing, or stream) or a jump followed by a local conversation.

Systems such as Java [GM94], Safe Tcl [BR95], and Omniware [ATLLW96], are concerned with the safe execution of untrusted code fragments. Safe Tcl is limited to Tcl scripts but Java and Omniware can work with any program (as long as the program is compiled into the bytecodes of the appropriate virtual machine). These three systems do not directly support mobile agents, although there are some very recent Java-based transportable-agent systems[1] [2] [3].

The best-known mobile-agent system is Telescript from General Magic [Whi94]. Telescript supports mobile computers and is used primarily on Personal Digital Assistants (PDA) such as the Sony Magic Link. The details of how Telescript agents jump between mobile hosts and handle disconnected operation are unclear. The Mobile Service Agent (MSA) system from ECRC [TLKC95] is another mobile-agent system supporting mobile computers, but it uses a less general mechanism than described in this paper. There are several other research projects that are building infrastructure for mobile agents. The most notable are Tacoma [JvS95], Itinerant Agents [CGH+95], Sodabot [Coe94], and ARA [Pei96]. New transportable-agent systems appear every day.[4]

Some database systems allow "stored SQL procedures" where you can define complex SQL commands and store them on the server [BP88]. The stored commands are executed at the server end during a user transaction. Some distributed file systems support disconnected operation, including Coda [KS92, MES95], Ficus [RHR+94], and others [HH95]. In these systems, applications on the laptop access the local file cache while the laptop is disconnected. On reconnection, the file system reconciles any differences with the appropriate file servers. The Bayou file system [TTP+95] internally uses a form of mobile code (but not agents) to handle reconciliation.

The Rover system [JdT+95] supports disconnected operation through queued RPC and relocatable dynamic objects (RDO). Queued RPC allows asynchronous RPC requests to be queued and then sent when the laptop connects; an asynchronous reply is delivered later. Relocatable dynamic objects (RDO) allow objects (code and data) to be downloaded from the server into the client, where they can execute closer to the user and, potentially, while disconnected. These RDOs are not true mobile agents because they do not move after they have begun execution.

Noble et al. [NPS95] describe the Odyssey system, in which applications on mobile computers can request upcalls whenever a change in resource state, such as network bandwidth or battery power, exceeds some threshold. This feature enables applications on mobile computers to change their behavior according to their environment, and would be a helpful substrate for an agent system.

There are of course many papers on mobile IP and packet forwarding. Perhaps the best background source is [Joh95]. Although a mobile-IP system would eliminate the need for our docking system to handle address changes, it would not provide the same support for disconnected operation, which is the primary benefit of our docking system.

4 Status

Agent Tcl has been publically released and is in active use at several sites in various information-management applications. The public version provides migration, communication, and access to the local screen and disk. Our internal version includes preliminary implementations of all of the support services described above (except for the agent-composition tool). The RPC mechanism, navigation services, and security mechanisms are the most complete. We are currently testing and evaluating these implementations. More information about Agent Tcl and our current research can be found at `http://www.cs.dartmouth.edu/~agent/`. A programmer's view of the system, including source code on CD-ROM, will be published soon [GKCR96].

---

[1] `http://www.ibm.co.jp/trl/projects/aglets/`

[2] `http://ptolemy.eecs.berkeley.edu/~wli/group/java2go/java-to-go.html`

[3] http://www.ftp.com/cyberagents/

[4] `http://www.cs.umbc.edu/agents/technology/asl.shtml`

References

[Age94] Intelligent agents. *Communications of the ACM*, 37(7):18–147, July 1994. Special issue.

[ATLLW96] Ali-Reza Adl-Tabatabai, Geoff Langdale, Steven Lucco, and Robert Wahbe. Efficient and language-independent mobile programs. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 127–136, 1996.

[BN84] Andrew D. Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, February 1984.

[BP88] Andrea J. Borr and Franco Putzolu. High performance SQL through low-level system integration. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 342–349, 1988.

[BR95] N. S. Borenstein and M. Rose. Safe Tcl. Available at `ftp://ftp.fv.com/pub/code/other/safe-tcl.tar.Z`, 1995.

[CGH+95] David Chess, Benjamin Grosof, Colin Harrison, David Levine, Colin Parris, and Gene Tsudik. Itinerant agents for mobile computing. Technical Report RC 20010, IBM T. J. Watson Research Center, March 1995. Revised October 17, 1995.

[Coe94] Michael D. Coen. SodaBot: A software agent environment and construction system. In Yannis Labrou and Tim Finin, editors, *Proceedings of the CIKM Workshop on Intelligent Information Agents, Third International Conference on Information and Knowledge Management*, Gaithersburg, Maryland, December 1994.

[Fal87] Joseph R. Falcone. A programmable interface language for heterogeneous distributed systems. *ACM Transactions on Computer Systems*, 5(4):330–351, November 1987.

[Gil96] Mark Giles. Navigation for transportable agents. Senior Honors Thesis, Dartmouth College Computer Science, June 1996.

[GKCR96] Robert Gray, David Kotz, George Cybenko, and Daniela Rus. Agent Tcl. In William Cockayne and Michael Zyda, editors, *Itinerant Agents: Explanations and Examples with CD-ROM*. Manning Publishing, 1996. Imprints by Manning Publishing and Prentice-Hall. To appear in late 1996.

[GKN+96] Robert Gray, David Kotz, Saurab Nog, Daniela Rus, and George Cybenko. Mobile agents for mobile computing. Technical Report PCS-TR96-285, Dept. of Computer Science, Dartmouth College, May 1996. Submitted to ACM MobiCom '96.

[GM94] James Gosling and Henry McGilton. The Java language: A white paper. Sun Microsystems, 1994.

[Gra95] Robert S. Gray. Agent Tcl: A transportable agent system. In *Proceedings of the CIKM Workshop on Intelligent Information Agents, Fourth International Conference on Information and Knowledge Management (CIKM 95)*, Baltimore, Maryland, December 1995.

[Gra96] Robert S. Gray. Agent Tcl: A flexible and secure mobile-agent system. In *Proceedings of the 1996 Tcl/Tk Workshop*, July 1996.

[HH95] L. B. Huston and P. Honeyman. Partially connected operation. *Computing Systems*, 8(4):365–379, Fall 1995.

[JdT+95] Anthony D. Joseph, Alan F. deLespinasse, Joshua A. Tauber, David K. Gifford, and M. Frans Kaashoek. Rover: A toolkit for mobile information access. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 156–171, December 1995.

[Joh95] D. B. Johnson. Scalable support for transparent mobile host internetworking. *Wireless Networks*, 1:311–321, October 1995.

[JvS95] Dag Johansen, Robbert van Renesse, and Fred B. Schneider. Operating system support for mobile agents. In *Proceedings of the Fifth Workshop Hot Topics in Operating Systems (HotOS)*, pages 42–45, May 1995.

[KS92] James J. Kistler and M. Satyanarayanan. Disconnected operation in the Coda file system. *ACM Transactions on Computer Systems*, 10(1):3–25, February 1992.

[LO95] Jacob Y. Levy and John K. Ousterhout. A Safe Tcl toolkit for electronic meeting places. In *Proceedings of the First USENIX Workshop on Electronic Commerce*, pages 133–135, July 1995.

[MES95] Lily B. Mummert, Maria R. Ebling, and M. Satyanarayanan. Exploiting weak connectivity for mobile file access. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 143–155, December 1995.

[NCK96] Saurab Nog, Sumit Chawla, and David Kotz. An RPC mechanism for transportable agents. In preparation. Expanded version currently available as Dartmouth PCS-TR96-280, March 1996.

[NPS95] Brian B. Noble, Morgan Price, and M. Satyanarayanan. A programming interface for application-aware adaptation in mobile computing. *Computing Systems*, 8(4):345–363, Fall 1995.

[Ous94] John K. Ousterhout. *Tcl and the Tk Toolkit*. Addison-Wesley, Reading, Massachusetts, 1994.

[Pei96] Holger Peine. The ARA project. WWW page `http://www.uni-kl.edu/AG-Nehmer/Ara`, Distributed Systems Group, Department of Computer Science, University of Kaiserlautern, 1996.

[RGK96] Daniela Rus, Robert Gray, and David Kotz. Autonomous and adaptive agents that gather information. In *AAAI '96 International Workshop on Intelligent Adaptive Agents*, August 1996. To appear.

[RHR+94] Peter Reiher, John Heidemann, David Ratner, Greg Skinner, and Gerald Popek. Resolving file conflicts in the Ficus file system. In *Proceedings of the 1994 Summer USENIX Conference*, pages 183–195, 1994.

[SG90] James W. Stamos and David K. Gifford. Remote execution. *ACM Transactions on Programming Languages and Systems*, 12(4):537–565, October 1990.

[Sto94] Alexander D. Stoyenko. SUPRA-RPC: SUbprogram PaRAmeters in Remote Procedure Calls. *Software—Practice and Experience*, 24(1):27–49, January 1994.

[TLKC95] Bent Thomsen, Lone Leth, Frederick Knabe, and Pierre-Yves Chevalier. Mobile agents. ECRC external report, European Computer-Industry Research Centre, 1995.

[TTP+95] Douglas B. Terry, Marvin M. Theimer, Karin Petersen, Alan J. Demers, Mike J. Spreitzer, and Carl H. Hauser. Managing update conflicts in a weakly connected replicated storage system. In *Proceedings of the Fifteenth ACM Symposium on Operating Systems Principles*, pages 172–183, December 1995.

[Whi94] James E. White. Mobile agents make a network an open platform for third-party developers. *IEEE Computer*, 27(11):89–90, November 1994.