Mobile agents for mobile computing

Robert Gray
David Kotz
Saurab Nog
Daniela Rus
George Cybenko

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

Technical Report PCS-TR96-285

May 2, 1996

Abstract

Mobile agents are programs that can move through a network under their own control, migrating from host to host and interacting with other agents and resources on each. We argue that these mobile, autonomous agents have the potential to provide a convenient, efficient and robust programming paradigm for distributed applications, particularly when partially connected computers are involved. Partially connected computers include mobile computers such as laptops and personal digital assistants as well as modem-connected home computers, all of which are often disconnected from the network. In this paper, we describe the design and implementation of our mobile-agent system, Agent Tcl, and the specific features that support mobile computers and disconnected operation. These features include network-sensing tools and a *docking* system that allows an agent to transparently move between mobile computers, regardless of when the computers connect to the network.

1 Introduction

Mobile computers have become increasingly prevalent as professionals discover the benefits of having their electronic work available at all times. Developing distributed applications that make effective use of networked resources from a mobile platform, however, is difficult for several reasons. First, mobile computers do not have a permanent connection into the network and are often disconnected for long periods of time. Second, when the computer is connected, the connection often has low bandwidth and high latency and is prone to sudden failure, such as when a physical obstruction blocks the signal from a cellular modem. Third, since the computer may be forced to use different transmission channels depending on its physical location, the performance of its

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

network connection can vary dramatically from one session to another. Finally, depending on the nature of the transmission channel, the computer might be assigned a different network address each time that it connects. In short, any distributed application that works on a mobile platform must deal with unforgiving network conditions.

In this paper we describe a system that uses *mobile agents* to support distributed applications for mobile computers. An *agent* is a program that is autonomous enough to act independently even when the user or application that launched it is not available to provide guidance and handle errors. A *mobile agent* is an agent that can move through a heterogeneous network under its own control, migrating from host to host and interacting with other agents and resources on each, typically returning to its home site when its task is done. We argue that mobile agents are a good paradigm for distributed applications and an *excellent* paradigm when mobile computers are involved.

We briefly describe a mobile-agent system, Agent Tcl, that is under development at Dartmouth College, and then present a system of support agents that provide network sensing and routing services. These support agents allow an agent to transparently migrate between a mobile computer and a permanently connected machine, or between one mobile computer and another, regardless of when the mobile computers connect to the network. These support agents provide a more general solution to mobile computing than approaches in which mobile agents are used simply to move an application onto a laptop for continued interaction with the laptop's owner.

The remainder of this section describes the rationale behind mobile agents and applications of mobile agents. Section 2 highlights related work. Section 3 gives an overview of the Agent Tcl system. Section 4 presents the agents that support mobile computing, our implementations of these agents, and an example sales application in which these agents may be used. Finally, Section 5 discusses our results and future work.

1.1 Why mobile agents?

Mobile agents are an effective paradigm for distributed applications, and are particularly attractive for partially connected computing. Partially connected devices include physically mobile computers such as laptops and personal digital assistants as well as home and business computers that are occasionally connected to the network over a SLIP or PPP modem connection. All of these devices are frequently disconnected from the network for long periods of time, often have low-bandwidth, unreliable connections into the network, and often change their network address with each reconnection. Mobile agents directly address the first two problems, and with low-level support, can handle the third problem without difficulty.

A mobile agent, for example, can migrate off a laptop and roam the Internet to gather information for its user. It can access the needed resources efficiently since it moves to their network location rather than transferring multiple requests and responses across the low-bandwidth laptop connection. Since it is not in continuous contact with the laptop, the agent is not affected by sudden loss of connection, and can continue its task even if the user powers down or disconnects from the network. When the user reconnects, the agent returns to the laptop with the result of its travels. Conversely, an application that lives in the network can send a mobile agent onto the laptop. The agent acts as the application's surrogate, interacting with the user efficiently and continuing to interact even in the event of long-term disconnection [TLKC95, JdT$^+$95].

Mobile agents also ease the development, testing and deployment of distributed applications since they hide the communication channels but not the location of the computation [Whi94b]; they eliminate the need to detect and handle network failure except during migration; they do not require the preinstallation of application-specific software at each site (although the agent system must be present); and they can dynamically distribute and redistribute themselves throughout the network. Mobile agents move the programmer away from the rigid client-server model to the more flexible peer-peer model in which programs communicate as peers and act as either clients or servers depending on their current needs [Coe94]. Mobile agents lead to more scalable applications since work can be *easily* moved to whichever network location is most appropriate. Mobile agents allow ad-hoc, on-the-fly applications that represent would be unreasonable investment of time if code had to be installed on each network site rather than dynamically dispatched. Finally, our experience with agent programming suggests that mobile agents are easier to understand than many distributed computing paradigms.

1.2 Applications of mobile agents

It can be argued that mobile agents are not an enabling technology since there are few applications (if any) that are *impossible* without mobile agents [HCK95]. However, the advantages of mobile agents lead to improved performance in many distributed applications, where performance is a matter of network utilization, completion time, programmer convenience, or just the ability to continue interacting with a user during network disconnection. Mobile agents are best viewed as a general tool for realizing arbitrary distributed applications. This view is reflected in the range of applications in which mobile agents are used.

Perhaps the most common examples of mobile code are Java applets. Java applets are interactive applications that can be dynamically pulled across the network with a Java-enabled WWW browser [Sun94]. Java applets are not true mobile agents since they migrate only once, before they start executing, and then only when requested by a user. Java applets are a powerful argument for mobile code, however, since most applets would be intolerably slow if they controlled the screen from a remote location. By moving to the local machine, an applet can control the screen efficiently without the need for pre-installation. Applets represent a special case of mobile agents. Mobile agents are much more powerful since they migrate at will.

True mobile-agent systems include Telescript [Whi94a, Whi94b], Tacoma [JvS95], Mobile service Agents (MSA) [TLKC95], and our own Agent Tcl [Gra95, Gra96]. Telescript agents are currently used for network management, active e-mail, electronic commerce, and business process management. In network management, a Telescript agent might carry a software upgrade onto a machine along with the code to perform the installation; the agent executes the installation code and disappears. In electronic commerce, a Telescript agent might leave a laptop, search multiple electronic catalogs on behalf of its user, and then return to the laptop with the best purchase price. The most visible use of Tacoma is StormCast, a system for distributed weather simulation in which the volumes of data are so immense as to make data movement impractical. Mobile Service Agents (MSA) have been used primarily in "follow-me" computing in which an application moves to the location of the user. One MSA demo involves an electronic conference proceedings. When a user connects his laptop to the conference's machines, an agent is sent to the laptop. The user interacts with the proceedings via this agent and can continue interacting even when disconnected.

Agent Tcl has been used primarily in information-retrieval applications. One information-retrieval application involves searching distributed collections of technical reports; another, medical records [Wu95]; and a third, three-dimensional drawings of mechanical parts [CBC96]. The advantages of agents in these retrieval applications is that each distributed collection can provide low-level primitives rather than all possible search operations; an agent can combine the primitives into efficient, multi-step searches. With the service agents for mobile computing that are introduced in Section 4, these same applications work unchanged on roving devices. Agent Tcl is also being used in workflow applications, in which an agent carries a multi-step task description from one site to another, interacting with the user at each site in order to carry out that user's part of the task [CGN96]. In Section 4, we describe a workflow application that involves both fixed and mobile computers, and that is supported easily with our mobile computing infrastructure. In this application, an independent traveling salesperson carries a laptop when visiting customers and uses software that helps to select vendors and products and to place orders. Agents represent orders and travel to the corporation's computers where they interact with billing, inventory, and shipping agents to arrange for the purchase. Agents are also used to explore vendor catalogs and search for products that meet the customer's needs. In all cases, the agents can function while the salesperson's laptop is disconnected.

2 Related work

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.

Systems such as Java [Sun94], Safe Tcl [BR], and Omniware [Col95] 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, but they address the same security issues and can be used as components in a larger system. Safe Tcl, for example, is used in Agent Tcl.

The best-known mobile-agent system is Telescript from General Magic [Whi94b, Whi94a]. 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] also supports 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]. As yet, however, none of these projects have considered mobile platforms.

Others have specifically suggested using mobile agents in mobile-computing environments. Pitoura and Bhargava propose a framework for agents to interact with heterogeneous mobile databases, but they focus on database consistency issues more than communication and transport issues [PB95].

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]. Other examples include [BZCS96], [IG93] and [WYOT93]. The idea is generally to allow a mobile computer to retain the same IP address regardless of location, so that applications on the laptop may continue to communicate with applications elsewhere. While such a system would simplify our laptop-docking scheme, since the laptop would never change address, it would not solve the primary problem of disconnection. Athan and Duchamp [AD93] go further in routing all of a laptop's communication through an "agent" that can filter data according to current network conditions, or store messages for delayed delivery.

3 Agent Tcl

Agent Tcl [Gra95, Gra96] is a mobile-agent system that we are developing at Dartmouth College and using in several information-management applications. Agent Tcl meets four main goals:

- Reduce migration to a single instruction like the Telescript *go* instruction [Whi94b], allow the instruction to appear at arbitrary points, and once the instruction is called, transparently capture the current state of the agent and transmit this state to the destination machine. The programmer should not have to explicitly collect state information. The system should handle all transmission details, including the possibility of the destination machine being disconnected or having a new network address.

- Provide transparent communication among agents. The communication primitives should be flexible and low-level, but should work the same regardless of whether the agents are on the same or different machines, and should hide all transmission details.

- Provide a simple scripting language as the main agent language, but support multiple languages and transport mechanisms, and allow the *straightforward* addition of a new language or transport mechanism.

- Provide effective security in the uncertain world of the Internet.

The architecture of Agent Tcl is shown in Figure 1. The agent server keeps track of the agents that are running on its machine, provides inter-agent communication facilities, accepts and authenticates agents arriving from other hosts, and restarts these agents in their own interpreter. All other services are provided by *agents*. Such services include navigation, network sensing, and access control. The agents themselves are separate processes executing the appropriate language interpreter. Each interpreter has the capability to capture the agent's state and send the state to a remote agent server.

The only language that we currently support is Tcl, although work on Java is underway. Tcl is a high-level scripting language that was developed in 1987 and has enjoyed enormous popularity [Wel95]. Tcl is an attractive agent language due to its simplicity, ease of use, and portability. A set of special commands was added to Tcl to create Agent Tcl. An agent uses these commands to migrate from machine to machine and to communicate with other agents. The most important

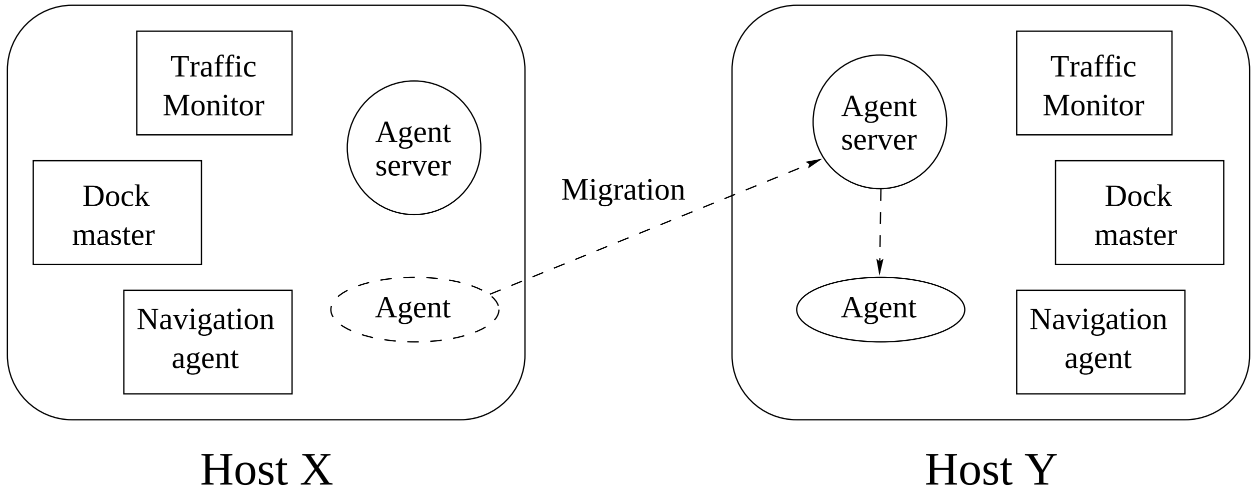

Figure 1: The server-based architecture of Agent Tcl. The agent server coordinates the activities of all local agents and accepts new agents that are arriving from other machines. All other services are provided by specialized agents such the as the *dock master*, *traffic monitor*, and *navigation* agents.

command is `agent_jump`, which migrates an agent from one machine to another. The `agent_jump` command captures the internal state of the agent, encrypts and digitally signs the state image, and sends the state image to the agent server on the destination machine. The server authenticates the agent and starts a Tcl interpreter. The Tcl interpreter restores the state image and resumes agent execution at the statement immediately after the `agent_jump`. Further details about Agent Tcl can be found in [Gra95] and on our web page.[1]. Details about Agent Tcl's security mechanisms can be found in [Gra96].

4 Mobile computing

Mobile agents are an excellent paradigm for implementing distributed applications, particularly in the context of partially connected computers. To be effective, however, the agent system must support disconnected operation in several ways.

- An agent must be able to jump off a partially connected computer (such as a laptop) and return to it later, even if the computer is only connected for brief periods and changes its address upon reconnection.
- An agent must be able to navigate through the Internet to find the services that it needs.
- An agent must be able to sense and react to the network environment, so that it may act autonomously while its user is disconnected.
- An agent must be able to communicate effectively with other agents.

In this section we describe our solutions, using "laptop" to mean any partially connected computer. Although our description and implementation are specific to the Agent Tcl system, the concepts are all generally applicable.

[1] `http://www.cs.dartmouth.edu/~agent`

4.1 Support for disconnected operation

Unlike traditional client-server computing, agents continue to operate even when the laptop is disconnected. 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* pairs each laptop with a permanently connected *dock* machine (Figure 2). While not all machines act as docks, all machines have a *dock-master* agent (Figure 1).

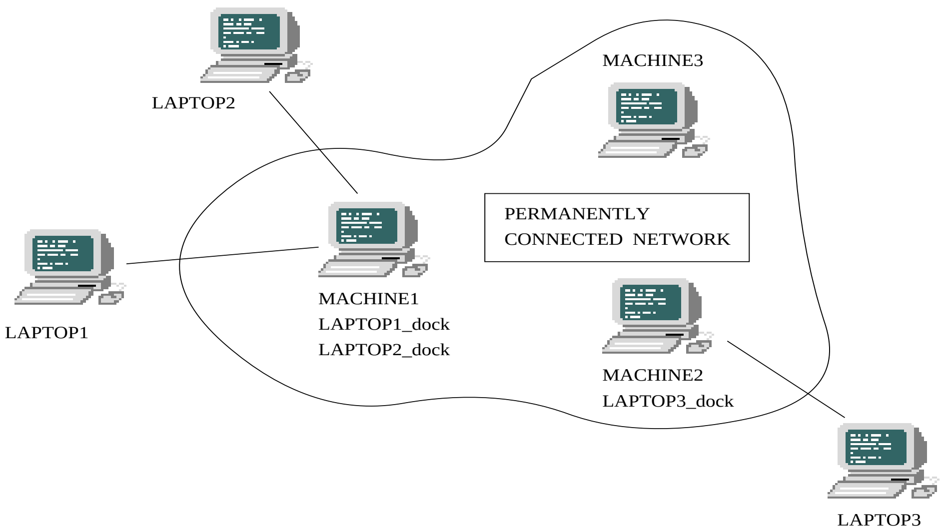

Figure 2: Laptop-docking system

Consider an agent wishing to jump to a disconnected laptop named D (Figure 3). 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.

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 3). This scheme has several advantages: the agents leave the laptop as soon as

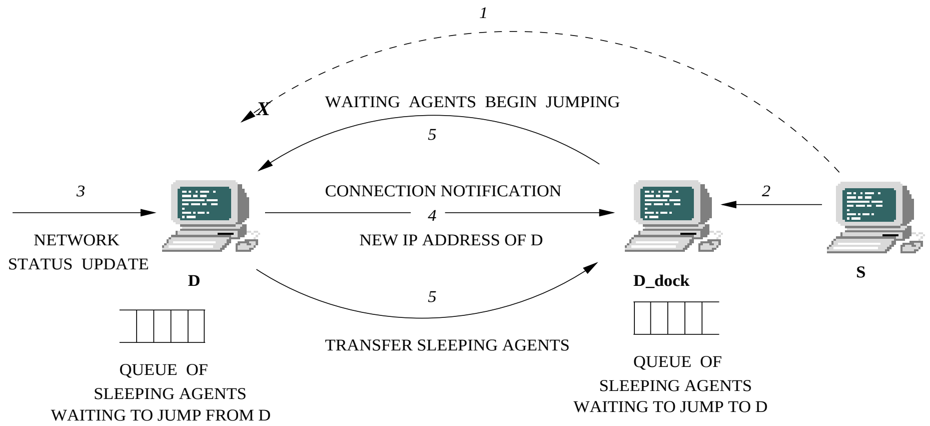

Figure 3: Jumping to or from the laptop

possible; agents do not miss any opportunities to leave; waiting 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.

Now consider a more complex case, where an agent's source (host S) and destination (host D) are both laptops (Figure 4). It is easy to imagine that they may never both be connected at the same time, making a direct jump impossible. The agent's state is captured on S, and saved on S's disk until the dock-master detects a network attached to S. At that point S's dock-master attempts to transfer the agent to D; when that fails, it transfers the agent to D's dock (*D_dock*). If *D_dock* is unreachable, perhaps due to a temporary problem in the Internet, the S dock-master tries to transfer the agent to *S_dock*. If *S_dock* is also unreachable, the dock-master will try the entire process again at a later time. If *S_dock* is reachable, the agent is sent to *S_dock*. The dock-master on *S_dock* will periodically attempt to transfer the agent to either D or *D_dock*. The agent may reside at *D_dock* until D connects and notifies the dock-master at *D_dock* of the new location of D. Once at D, the agent's state is restored.

We are extending our *laptop docking system* to support multi-destination jumps, which are useful when an agent wishes to visit multiple hosts $(D_1, D_2, ..., D_n)$ but *in no particular order*. This situation arises when the agent is searching all of the sites for information, or when it needs to visit one of a replicated set of servers. The multi-destination jump allows the agent to travel in a manner most suitable to the present network conditions. The dock-master agent on S first tries to transfer the agent to one of the final destinations by trying each one in order $(D_1, D_2, ..., D_n)$. If all the destinations are unreachable, the S dock-master transfers the agent to *S_dock*. The dock-master at *S_dock* periodically tries to reach the destinations until one of the transfers succeeds. *S_dock* does not transfer the agent to a dock D_k*_dock* in order to avoid premature commitment to a destination that may rarely connect, although this issue is a matter for further research. When the agent awakes (returns from its call to *agent_jump*), it knows that it has arrived at one of the destinations. A quick check of the local host name confirms the particular destination.

For agents that desire more control over the jumping process, we provide hooks to allow agents to query the status of the network connection, to request a failure notification rather than being blocked when the jump destination cannot be reached immediately, or to request that the jump go

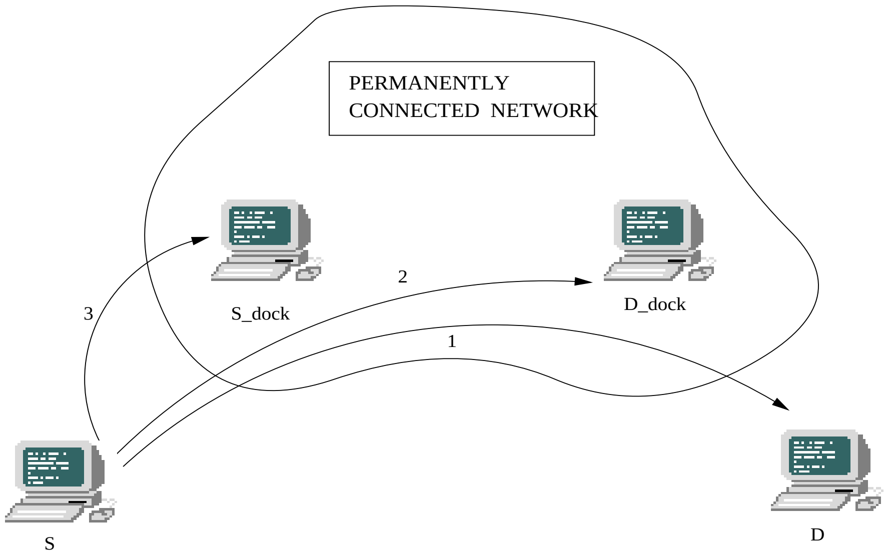

Figure 4: Laptop to laptop jump

as far toward the destination as possible and then wake up the agent.

4.2 Agent navigation and adaptation

The world of an agent is dynamic and uncertain. Machines go up and down, the information stored in repositories changes, and the exact sequence of destinations and steps needed to complete an information-gathering task often is not completely known at the time that the agent is launched into the world. An autonomous agent is crippled without external state (what the agent can perceive about the state of its world) since it has no way of perceiving and adapting to the dynamic changes in its environment. In this section we describe the sensors that allow an agent to determine its external state and a mechanism that uses these sensors for adaptive navigation.

Network sensing. Network sensing, at least the ability for a laptop to detect the state of its network connection, is an integral part of our *laptop docking system* described in the previous subsection. It performs an even more important task, however, when providing agents with information about the expected transit time across the network and about whether a network site is reachable at all. This information enables agents to adapt to changing network conditions. Consider an agent that needs to visit information resources at several sites. A smart agent should be able to adapt to the fact that some sites may currently be unreachable, and to visit other sites first. An even smarter agent may be able to plan a sequence of visits given an estimate of the current network delay to each site. Other agents may wish to tailor their behavior to the current bandwidth available, such as the amount or format of the data that they carry with them.

We provide a set of network sensing tools that the agents can use to gather information about the status of the network.

- A tool to determine whether the local host is physically connected. This tool "pings" the broadcast address on the local subnet; if there is any response in a short interval, the network is connected.

- A tool to determine whether a specific host is reachable; this is just the standard "ping."
- A tool to determine the expected bandwidth to a remote host, so that agents can choose their destination or amount of data based on the bandwidth. Rather than measuring the bandwidth by sending lots of data to the remote host, which would often take as much time as sending the agent itself, we attempt to predict bandwidth from experience. A *traffic monitor agent* at each site tracks information about all recent communications (bytes moved and time required), which is provided by the local agent server. Application agents contact the network monitor to obtain estimates of bandwidth or latency, which are computed from the recorded information. Our traffic monitor uses a weighted average of all communications with the requested remote site, weighting recent communications more heavily than older communications. If there are no recent communications with the requested site, the traffic monitor may use data from recent communications with "similar" sites, that is, other sites in the same subnet or domain as the requested site.

Navigation agents. To locate other agents that can serve their needs, agents need access to a dynamic index of service agents and their locations. We use a system of *virtual yellow pages* to help the agents decide where to go. These yellow pages contain listings of services and their locations. By consulting these navigation services and using their network sensing tools, agents can formulate adaptive navigation plans to visit some of the services.

The virtual yellow pages are a distributed database of service locations maintained by a hierarchical set of navigation agents. Services register with the navigation agents that are scattered throughout the system (Figure 5). Each machine has a specialist agent that knows the location of some of the navigation agents (which in turn know the locations of services and other navigation agents). In general, by consulting the local specialist agent and then visiting one or more navigation agents, an application agent can obtain the necessary list of services and their locations.

Since the information landscape changes, the virtual yellow pages are not static entities. We use adaptive learning methods to keep the virtual yellow pages up to date.

- New services register with one or more navigation agents to advertise their location. They describe their service through a list of keywords. For example, in Figure 5, service 1 registers with navigation agent 2 by the following protocol: service 1 first contacts the specialist agent on its machine which knows the location of navigation agent 2. Service 1 then sends a registration message to navigation agent 2 which adds service 1 to the database.
- An application agent locates a list of navigation agents by querying the specialist agent on the local host (Figure 5). The application agent then consults the navigation agents by providing a list of keywords. The navigation agent returns a list of matching services from its database. After visiting some of the services, the application agent revisits the navigation agents to provide feedback on which of the sites were useful and which were useless. These "consumer reports" enable the navigation agents to prioritize their lists.
- Agents that discover services accidentally report the corresponding sites to the navigation agents. For example, services relevant to one task may be discovered while handling a different but related task. Such a situation might arise if an agent handles textual queries about different topics; while finding documents relevant to one topic, it may discover document collections that relate to another. Alternatively, an agent might receive different site information from two navigation agents; it feeds the differences back to the navigation agents.

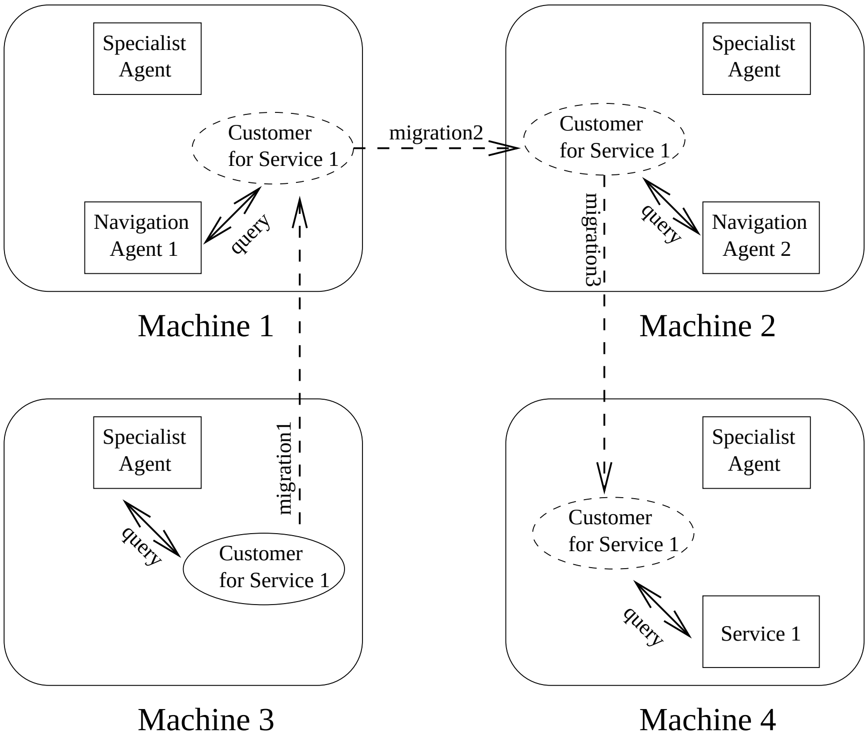

Figure 5: An example of navigation. Each machine has a number of agents running on it (denoted by rectangular blocks.) The *specialist agents* know about the location of one or more *navigation agents*. There are two navigation agents shown here: one on machine 1 and one on machine 2. The navigation agent on machine 2 knows about service 1, but the navigation agent on machine 1 does not. The specialist agent on machine 3 knows about both navigation agents. The *customer agent* on machine 3 uses the following protocol to locate service 1. It first contacts its local specialist agent and finds the location of navigation agents 1 and 2. Then it migrates to machine 1 and queries navigation agent 1 about service 1. This navigation agent does not know about service 1 since service 1 is only registered with navigation agent 2. The customer agent then migrates to machine 2 where it queries navigation agent 2 and finds the location of service 1. Finally, the customer agent migrates to the location of service 1.

Application agents construct an initial plan for accomplishing their task by using the prioritized list of services that they receive from the navigation agents. Most applications will want to visit either one or all of the sites on the list. Using the network-sensing tools, however, they may choose to skip some sites that are not reachable or to which there is a particularly slow connection, and then return to them later.

4.3 Inter-agent Communication

Agent Tcl currently provides two levels of agent communication. The low-level mechanisms allow agents to communicate through message passing or through a direct connection that is established when an agent issues the `agent_meet` command and the receiving agent accepts the meeting.

The higher-level Agent Remote Procedure Call (ARPC) [NCK96] mechanism builds on top of these primitives, adding structure as well as a higher-level abstraction to the communication. Server agents in the system register with the local "name-server" agent by specifying their interface in a flexible definition language. Client agents search for a service by providing a similar interface and having the "name-server" find a match from among its registered servers. This flexible interface-matching technique helps agents to communicate even when they share only a subset of a complex interface. For example, a server might have added non-standard features, or might have an older but upwardly-compatible interface.

4.4 Example

Returning to our example of the traveling salesperson, we see how the above infrastructure supports this distributed, mobile application. While on the road, the salesperson carries a laptop or PDA loaded with catalog and order-entry software. While at the customer's location, the software helps to select appropriate products and vendors, prepare a quote, and place an order. The software creates an agent for each order, which must be approved by the salesperson's supervisor before the order is submitted. The agents immediately try (and fail) to jump off of the salesperson's laptop to the supervisor's computer, and are queued by the dock-master to await the laptop's reconnection. After a day of customer visits, the salesperson connects the laptop to the network, and all of the agents jump off on their way to the supervisor's computer. The laptop need be connected for only a few seconds.

If the supervisor is also a traveler, then the agents must reach the supervisor's laptop. If that laptop is not connected, the agents wait at that laptop's dock until the laptop reconnects. The agents ask the supervisor to examine and approve the orders, and then they continue on their way to the appropriate vendors (perhaps after another delay to wait for the laptop to reconnect, and perhaps forking into multiple agents, one for each vendor).

Once at the vendor, an order agent interacts with the vendor's billing agents to record the sale for billing purposes, with inventory agents to determine which items are in stock, and with shipping agents to arrange shipping. In each interaction, the agent may use customized code to adapt to price changes, discontinued or back-ordered items, and shipping details.

Eventually, the order agent returns to the salesperson's laptop to inform them that the sale was complete, and whether shipping was successful.

In this application, several of the computers are inherently mobile and disconnected, so the agents must depend on the dock-masters to help them jump from machine to machine. The use of agents allows for considerable flexibility. Through standard protocols, the vendors and independent salespeople can use software produced by different third-party vendors, which compete on the basis of other features. In particular, the salesperson chooses an order-placement software package according to its ability to produce adaptive order agents; since the order agents are executable code, they can implement adaptive strategies that may not have been anticipated by the writers of the vendor software. While it is possible to build a traditional system with fixed interfaces that exchange data only, only mobile agents can allow this kind of flexible innovation.

5 Discussion

We validated our system in our labs through an experiment with a laptop computer called *Bond*.[2] We started an agent on Bond, and the agent immediately jumped off Bond to interact with a remote server. Before it could return, we disconnected Bond, carried it to another lab, connected it to a different subnet, and reconfigured it with a new IP address.

[2] James Bond.

Meanwhile, the traveling agent had finished its task and had attempted to jump back to Bond. The jump failed, so it was transferred to Bond_dock, where its state was saved on disk.

When Bond reconnected at the new address, its dock-master discovered the new connection and new address, and sent a message to Bond_dock, back in the original lab. Bond_dock then sent the waiting agent on to Bond. We then repeated the experiment, carrying Bond back to its original address.

This simple experiment demonstrates how our mobile-agent system supports mobile computing in that an agent was able to leave the laptop and return home twice, despite disconnection, reconfiguration, and reconnection at a different IP address.

Our system still has a few limitations, however:

1. If an agent is running on a machine when the machine goes down, the agent is lost.

2. If an agent is running on a machine and the machine becomes disconnected from the network for a long period of time, the agent remains in exile on this machine for the entire time.

3. Currently, a laptop dock-master agent monitors the state of the local network connection through periodic "pings" to the broadcast address on the local subnet. If the laptop is connected for less time than the interval between pings, the dock-master will not detect the connection. A better solution is to obtain an interrupt directly from the operating system when the network connection changes [NPS95].

4. Through a simple convention, it is easy to locate the dock for a given host: the dock for host named X.domain is the host named *X_dock*.domain. There are some environments, however, that include nameless hosts, most commonly, personal computers assigned an IP address dynamically at boot time. Our system cannot currently accommodate nameless hosts.

In developing the tools for agent support of mobile computing, we have found that the operating system infrastructure available to us limited the possible solutions. Specifically, the following low-level operating systems features would enable more elegant solutions:

1. As mentioned above, we could avoid a busy-wait sensor for network connectivity if the operating system could provide a flag or an interrupt every time the local network connection goes up or down.

2. Network routers, and some hosts, have information about network connectivity and delays that allow them to route packets to their destination. If that information were made available to agents, we might be able to make much better predictions than those available from the traffic monitor agent.

Future work. There are many interesting areas for future work. As we mentioned, there are a few small operating-system extensions that would be helpful, and we are investigating multi-destination jump support. We plan to integrate our inter-agent message-passing with the docking system, so that messages go through docks when necessary. We are also refining our bandwidth-prediction tools. We are considering support for persistent storage, so that an agent may leave some of its data (such as the results of a database search) at one host, carry a small part of its data along with it, and yet be able to remotely access the saved data if necessary. Finally, we are developing the traveling-salesperson application as a real-world demonstration of our ideas; most of the pieces exist in simple forms and need to be extended and combined into the single application.

Summary. We have constructed a system for supporting mobile computing with mobile agents. We argue that mobile agents allow a range of adaptive, flexible applications in distributed heterogeneous systems with non-permanent network connections. We describe our experiences with using this system, and identify a few operating-system extensions that would enable efficient, reliable, and simple mobile computing support through mobile agents.

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 working prototypes of all of the support services described above. We continue to extend and evaluate these implementations. More information about Agent Tcl and our current research can be found on our WWW page.[3]

Acknowledgements

Many thanks to the students that have helped with the construction of the support agents described in this paper: Ting Cai, Saurab Nog, and Vishesh Khemani built the "dock" system; Dawn Lawrie and Mark Giles built the navigation system; David Hofer and Miranda Barrows built the network-sensing tools; and Saurab Nog and David Hofer maintained the Agent 007 research lab. Thanks also to the students of CS88/188 (Fall 1995) for their many discussions leading to these ideas. Finally, thanks to ONR and AFOSR for their generous funding.

References

[AD93] Andrew Athan and Dan Duchamp. Agent-mediated message passing for constrained environments. In *Proceedings of the Mobile and Location-Independent Computing Symposium*, pages 103–107, August 1993.

[BN84] A. D. Birrell and B. J. 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.

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

[BZCS96] Mary Baker, Xinhua Zhao, Stuart Cheshire, and Jonathan Stone. Supporting mobility in MosquitoNet. In *Proceedings of the 1996 Winter USENIX Conference*, pages 127–139, January 1996.

[CBC96] Kurt Cohen, Aditya Bhasin, and George Cybenko. Pattern recognition of 3D CAD objects: Towards an electronic yellow pages of mechanical parts. *International Journal of Intelligent Engineering Systems*, 1996. To appear.

[3] `http://www.cs.dartmouth.edu/~agent/`.

[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.

[CGN96] Ting Cai, Peter A. Gloor, and Saurab Nog. Dartflow: A workflow management system on the web using transportable agents. Technical Report PCS-TR96–283, Dept. of Computer Science, Dartmouth College, May 1996.

[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.

[Col95] Omniware technical overview. Colusa Software White Paper, 1995. Available from `http://www.colusa.com`.

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

[Gra95] Robert S. Gray. Agent Tcl: A transportable agent system. In James Mayfield and Tim Finin, editors, *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 Mark Diekhans and Mark Roseman, editors, *Proceedings of the Fourth Annual Tcl/Tk Workshop (TCL '96)*, Monterey, California, July 1996. To appear.

[HCK95] Colin G. Harrison, David M. Chess, and Aaron Kershenbaum. Mobile agents: Are they a good idea? Technical report, IBM T. J. Watson Research Center, March 1995.

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

[IG93] John Ioannidis and Gerald Q. Maguire, Jr. The design and implementation of a mobile internetworking architecture. In *Proceedings of the 1993 Winter USENIX Conference*, pages 491–502, January 1993.

[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.

[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. Technical Report PCS-TR96–280, Dept. of Computer Science, Dartmouth College, 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.

[PB95] Evaggelia Pitoura and Bharat Bhargava. A framework for providing consistent and recoverable agent-based access to heterogeneous mobile databases. *ACM SIGMOD Record*, 24(3):44–49, September 1995.

[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.

[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] J. Stamos and D. Gifford. Remote evaluation. *ACM Transactions on Programming Languages and Systems*, 12(4):537–565, October 1990.

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

[Sun94] The Java language: A white paper. Sun Microsystems White Paper, Sun Microsystems, 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.

[Wel95] Brent Welch. *Practical Programming in Tcl and Tk.* Prentice Hall, 1995.

[Whi94a] James E. White. Mobile agents make a network an open platform for third-party developers. *IEEE Computer*, 27(11):89–90, November 1994.

[Whi94b] James E. White. Telescript technology: The foundation for the electronic marketplace. General Magic White Paper, 1994.

[Wu95] Yunxin Wu. Advanced algorithms of information organization and retrieval. Master's thesis, Thayer School of Engineering, Dartmouth College, 1995.

[WYOT93] Hiromi Wada, Takashi Yozawa, Tatsuya Ohnishi, and Yasunori Tanaka. Mobile computing environment based on internet packet forwarding. In *Proceedings of the 1993 Winter USENIX Conference*, pages 503–517, January 1993.