Transportable Information Agents

Daniela Rus, Robert Gray, and David Kotz

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

Abstract

Transportable agents are autonomous programs. They can move through a heterogeneous network of computers under their own control, migrating from host to host. They can sense the state of the network, monitor software conditions, and interact with other agents or resources. The network-sensing tools allow our agents to adapt to the network configuration and to navigate under the control of reactive plans. In this paper we describe the design and implementation of the navigation system that gives our agents autonomy. We also discuss the intelligent and adaptive behavior of autonomous agents in distributed information-gathering tasks.

1 Introduction

Modern information systems have data distributed over heterogeneous and unreliable networks. We wish to develop sophisticated methods for browsing, searching, and organizing distributed information systems. Traditional approaches to distributed information access co-locate the data and the computation needed to process it by bringing the data to the computation. We advocate a novel approach that brings the computation to the data in the form of *transportable agents*. A transportable agent is a program that can migrate from machine to machine in a heterogeneous network. Transportable agents have navigation autonomy, that is, they are capable of traveling freely and independently throughout a computer network. This approach requires an agent to have substantial intelligence in making decisions and filtering information.

In this paper we discuss our transportable-agent system called *Agent Tcl* and describe a distributed information-gathering experiment in a network of mobile computers, such as laptops. Mobile computers do not have a permanent connection into the network and are often disconnected for a long period of time. We focus on sensori-computational aspects of the system that allow the agents to observe changes in their world and to navigate adaptively through a network, guided by reactive plans. For example, a user might write a transportable agent for a distributed information-gathering task, launch it from a laptop connected to the Internet in California, and disconnect the laptop. The agent will navigate the Internet autonomously, gathering and organizing information. Some time later, the user might resurface on the Internet in New York, where the laptop is assigned a different IP address. The agent should detect the presence of the laptop on the Internet at the new location and return to it with the search results. The system we describe here permits the quick specification of adaptive autonomous agents for such classes of tasks.

Transportable agents navigate heterogeneous networks under the control of reactive plans that give *adaptation* powers to these agents. We support adaptation with an infrastructure of *network-sensing* modules. Agents can sense hardware conditions (for example, whether a host is connected to the network) or software conditions (for example, a specific change in a database). The systems infrastructure for information processing on mobile computers is described in detail in [GKNRC95].

Transportable agents provide a convenient, effi-

This paper describes research done in the Dartmouth Transportable Agents Laboratory at Dartmouth. This work is supported in part by the Navy and Air Force under contracts ONR N00014-95-1-1204 and AFOSR F49620-93-1-0266. Mark Giles and Dawn Lawrie implemented the virtual yellow-page system. David Hofer and Saurab Nog implemented the network sensors. Katya Pelekhov implemented the Smart server.

Int'l Conference on Autonomous Agents, Feb. 1997, doi:10.1145/267658.267721.

cient, robust, and intelligent paradigm for implementing distributed applications, especially in the context of wireless computing. First, by migrating to the location of an electronic resource, an agent can access the resource locally and eliminate costly data transfers over congested networks. This reduces network traffic, because it is often cheaper to send a small agent to a data source than to send all the intermediate data to the requesting site. Second, the agent does not require a permanent connection to the host machine (*e.g.*, the computer from where an agent is launched). This capability supports distributed information-processing applications on mobile computers. Third, the network-sensing capabilities enable agents to autonomously find the host computer, even when the host changes its geographical location. Our system infrastructure for using mobile agents with mobile computing is described in [GKNRC95]. Fourth, the network software- and hardware-sensing capabilities permit transportable agents to navigate adaptively. Fifth, our transportable agents can communicate with each other even when they do not know their specific locations in the network. Finally, agents have autonomy in decision making: by using feedback from visiting a site, they can independently modify the overall plan or refine ill-specified queries. When combined with communication, decision-making enables our agents to be negotiators. Agent Tcl supports negotiation through an infrastructure of electronic cash, transactions on electronic cash, arbitration on electronic cash transactions, and economic policies for resource control.

Mobile agents provide a simple, adaptive, and unified solution for networking mobile computers and for supporting many distributed systems applications. A good transportable agent system eliminates the need for application-specific solutions, while providing similar performance. Specifically, with such an agent system there is (1) no need for high-level search engines at the remote sites (*e.g.*, the search application); (2) no need for automated installations (*i.e.*, follow-me computing[1]) and (3) no need for distributed applications to build their own control language (*e.g.*, programmable distributed databases.)

2 Previous Work

Kahn's proposal [KC86] about architectures for retrieving information from electronic repositories was the first recognition of the utility of software agents for information processing. It provides context for the issues discussed in this paper. We draw from research results in several distinct areas: operating systems, agents, information retrieval, and mobile robotics.

Although little has been published on transportable agents, much work has been done concerning the general concept of remote computation. Remote Procedure Call (RPC) [BN84] was an early form of remote client-server processing. Falcone [Fal87] discusses a distributed-system in which a programming language provides a remote service interface as an alternative to RPC calls. Stamos and Gifford [SG90] introduce the concept of Remote Evaluation (REV), in which servers are viewed as programmable processors. The Telescript technology introduced by General Magic, Inc. in 1994 was the first commercial description of transportable agents [Whi94]. Prototypes of transportable agent systems exist [KK94, Gra95, Gra96, JRS95, TLKC95]. Telescript has been primarily used in connection with Personal Digital Assistants, network management, active email, electronic commerce, and business process management. The details of how Telescript agents jump between mobile hosts and handle disconnected operations are unclear. The Mobile Service Agent [TLKC95] has been used primarily for "follow-me" computing, and it is less general than our system.

In the software-agents literature, much time and effort has been devoted to designing task-directed agents and to the cognitive aspects of agents. Agents are called *knowbots* by [KC86], *softbots* by [EW94], *sodabots* by [KSC94], *software agents* by [GK94], *personal assistants* by [Mae94, MCF94], and *information agents* by [RS93, RS96]. We are interested in the same class of tasks as [EW94, Mae94, MCF94, KSC94]. Etzioni and Weld [EW94] use classical AI planning techniques to synthesize agents that are Unix shell scripts. Mitchell and Maes [MCF94, Mae94] study the interaction between users and agents and propose statistical and machine-learning methods for building user models to control the agent actions. Rus and Subramanian [RS93, RS96, RS96] propose a modular, open, and customizable agent architecture organized around a notion of structure recognition. In our previous work [Gra95, Gra96, NCK96, GKNRC95, KGR96] we describe other aspects of Agent Tcl.

The analogy between mobile robots in unstructured physical environments and information agents in rich multi-media environments is not just metaphorical. We have observed that the lessons learned in designing task-directed mobile robots [Bro86] can be imported to the problem of distributed information gathering with transportable agents. We also draw from recent

[1] In follow-me computing, a user's applications are sent to her current location so that she may interract with them more effectlively.

results in analyzing the information requirements for robot tasks [DJR93].

3 Transportable Agents

Autonomous agents should move independently. A *transportable agent* is a program that can migrate under its own control from machine to machine in a heterogeneous network. In other words, the program can suspend its execution at an arbitrary point, transport to another machine, and resume execution on the new machine. Transportability is a powerful attribute for information-gathering agents since their world is usually a distributed collection of information resources, each of which can contain tremendous volumes of data. By migrating to the network location of an electronic resource, a transportable agent eliminates all intermediate data transfer and can access the resource efficiently even if the resource provides only low-level primitives for working with its contents. This benefit is particularly great with a low-bandwidth network connection for which moving the data is often infeasible; moving the computation to the data with a transportable agent is a convenient and efficient alternative.

Before transportable agents can be used effectively, several challenges must be met. Most difficulties arise from the fact that we are allowing code to roam at will through a distributed system. The most important issues are to protect machines from malicious agents and agents from malicious machines; to provide effective fault tolerance in the uncertain world of the Internet; to allow programmers to write and debug agents quickly and easily; to make agents almost as efficient as highly tuned, application-specific servers; and to provide a location-independent namespace in which agents can communicate. We are currently addressing these issues.

3.1 Agent Tcl: a system for transportable agents

Agent Tcl [Gra95] reduces migration to a single instruction, provides simple communication among agents, supports multiple languages and transport mechanisms, runs on generic platforms, and provides effective security, fault tolerance and performance. In the current implementation, agents are written in a modified version of the Tool Command Language (Tcl) [Ous94]. Tcl is a high-level scripting language and is an attractive agent language since it is highly portable, easy to use, and easy to make secure (due to the large amount of existing work that addresses the problem of executing a Tcl program from an untrusted source). Our modified version of Tcl is the same as standard Tcl except that the internal state of an executing script (the stack, the contents of variables, etc.) can be captured at an arbitrary point. In addition, the modified version of Tcl provides a special set of commands that allow a Tcl script to migrate and to communicate with other migrating scripts.

Thus, one of our transportable agents is simply a Tcl script that runs in the modified Tcl interpreter and uses the agent commands to roam through a network and interact with other agents. A Tcl script can decide to move to a new machine at any time. It issues the `agent_jump` command, which suspends script execution, captures and packages the internal state of the script, and sends this state image to a *server* on the destination machine (a special server runs on every machine to which transportable agents can be *sent*). The server restores the state image and the Tcl script continues execution on the new machine from the exact point at which it left off. The Tcl scripts can communicate via message passing, or remote procedure call [NCK96]. An agent can use the Tk toolkit to present a graphical user interface on either its home machine or on a remote machine to which it has migrated. In addition, Agent Tcl includes the beginnings of a mechanism for considering economic issues in societies of agents. Agents interact by exchanging electronic cash. Their use of resources is limited by a set of policies for resource control.

4 Sensing

To remain efficient, agents unleashed in the network must operate without continuous contact with their home sites, without user intervention, and despite complications. For example, if the agent was launched from a mobile platform that has since become temporarily disconnected from the network, it must be prepared to proceed on its own rather than waiting an unknown amount of time for the mobile platform to reappear. Complications arise because agents operate in a dynamic and uncertain world. Machines go up and down, the information stored in repositories changes, and the exact sequence of steps needed to complete an information-gathering task is not completely known at the time the agent is launched into the world. Without external state (what the agent can perceive about the state of its world) an autonomous agent is crippled since it has no way of perceiving and adapting to the dynamic changes in its environment. This section elaborates on the "sensors" that allow an agent to discover important information about its environment and to establish its external state. We focus on the following three components of external state: hardware, software, and other agents.

4.1 Sensing the state of the network

Our agents can determine whether a network site is reachable and can predict the expected transit time across the network and the expected processing time at the site. This information allows an agent to adapt to currently unreachable or overloaded sites by visiting other sites first. Smart agents can use information about reachability, network delays, and available bandwidth to intelligently construct routing plans. We have implemented several network sensors:

Local connectivity. This sensor determines whether the local host is physically connected by "pinging" the broadcast address on the local subnet; if there is any response in a short interval, the network is connected.

Site Reachability. This sensor returns true if a specific site is reachable. This is implemented by using the Unix `ping` command.

Network Load. This sensor tests the expected bandwidth to a remote host. It predicts latency by consulting a local table that compiles traffic history information. The table estimates bandwidth for several time ranges: 0-15 seconds; 15-30 seconds; 30 seconds-3 minutes; 3 minutes-2 hours; 2 hours-1 day; 1 day-2 weeks; and 2 weeks and up. The table gets updated incrementally and each category is weighted differently in the computation for the estimate.

4.2 Sensing software changes

Agents are often faced with the problem that a resource is unavailable, does not contain the desired information, or is expected to contain additional relevant information at an unknown point in the future. Depending on the application, the agent might choose to report failure, move to an alternative resource, or wait for the desired resource or information to become available. Our agents use information-retrieval techniques to detect when the state of a software resource has changed. Significant activity on a resource is signaled by an increase in the resource size (detected by looking at the size) or a shift in content (detected by the information-retrieval methods we use in Section 7). Figure 1 shows an agent that monitors a set of files and directories and sends an email message when it senses significant activity on a file. The agent works by creating one child agent for each remote filesystem. Each child monitors one or more directories and sends a message to the parent when there is significant file activity. The parent then contacts the user's mail agent to send the message. Although simplistic, this agent illustrates the general task of waiting for an event to occur and then reacting appropriately, a task that is faced by nearly every agent.

5 Navigation

Agents implemented in Agent Tcl have the ability to move by themselves through a network. But where should they go? Agents need either a partial model or partial knowledge of both the task and the environment. We use an implicit scheme that provides a system of *virtual yellow pages* to help the agents decide where to go. These yellow pages contain listings of services and resources. By consulting these virtual yellow pages and using the network-sensing tools, an agent selects a list of services relevant for its task and formulates adaptive plans to visit some of the sites.

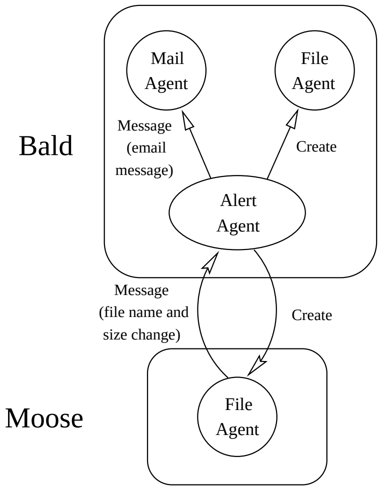
Virtual Yellow Pages. The virtual yellow pages are a distributed database of service locations maintained by a hierarchical set of navigation agents. Services register with the navigation agents that are scattered throughout the system (Figure 2) and manage the yellow pages. Each machine has a specialist 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.

Construction of Virtual Yellow Pages. 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 2, Service 1 first contacts the specialist agent on its machine to find the location of Navigation Agent 2. Service 1 then sends a registration message to Navigation Agent 2, which adds Service 1 to the database.

Locating Services. An application agent locates a list of navigation agents by querying the specialist agent on the local host (Figure 2). 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.

Adaptive selection of the best service. After visiting some of the services, the application agent revisits the navigation agents to provide feedback about the sites (speed of service and usefulness of results). These "consumer reports" enable the navigation agents to learn which services are most useful and to prioritize services accordingly.




```
set email_agent "bald rgray_email"      # machine and name of email agent
set machines "bald moose"
set directory "~rgray"

  # get a name from the server

agent_begin

  # submit the "file" agents that watch for changes in file size

for each m $machines {
  agent_submit $m -vars directory -proc file_watch {file_watch $directory}
}

  # wait for one of the "file" agents to send a message saying that a
  # file has changed size; then send an alert message to the user by
  # asking the user's email agent to send a message to its owner

while {1} {

  agent_receive code string -blocking
  set alert [construct_alert $string]
  agent_send $email_agent {SEND OWNER $alert}

}
```

Figure 1: The alert agent monitors a set of files and sends an email message to the user when it detects a significant file activity. A simplified version of this agent appears at bottom. The network location of the various agents is shown at top. `file-watch` looks at the size of the file and compares the content of the file against a query or a previous version of the file using information retrieval techniques [Sal91].

As a general policy for identifying the optimal service, we keep the average feedback of each of the service providers. In most cases the navigation agent should recommend the best service it knows. This method converges to the best service in a static system. We consider a dynamic system, where services appear and disappear[2] by augmenting the best-first policy with a method that encourages initial exploration of other agents. The exploration function returns an overly optimistic estimate of the usefulness of a service until the service is explored N times; after that the real average value of the agent is used for ranking. Figure 3(left) shows the performance of two exploration functions in a system where an initial virtual page consists of 5 services, not ranked in any particular order. We ran a simulation in which agents visited the services and returned with feedback on the goodness of the service. Several iterations (about 100) in this experiment, a new and better service (Agent 6) was added. We examined evaluation functions for several values on N. The case $N = 1$ is denoted by Avg, and $N = 5$ is denoted by $High$. Both cases converge to Agent 6.

This algorithm does not take into account that the relative usefulness of an agent may vary over time. One agent may improve on another's service, or it may become outdated or congested. To discover bad services that have radically improved their performance, a small randomization factor is added in the exploration function (see Figure 3(right)). Our experiments with dynamic service landscapes show that the best service is always found, although it may take on the order of 100 trials to converge to it.

Navigation Plans. Agents construct an initial plan, *e.g.*, an ordered sequence of sites, from the information provided by the virtual yellow pages. The agent uses this list to sequentially move from site to site, advancing when the necessary processing at the current site has been completed. The agent might also choose to launch child agents at certain points. The plan of the agent need not be static, however. The agent formulates and reformulates the plan by consulting its sensors and adapting on-line to changes in network configuration and software content. For example, if the plan consists of the sequence A, B, C, D and machine A is sensed to be down while B is sensed to be up, the agent greedily rearranges the sequence to B, A, C, D. Analogously, if the bandwidth to A is much lower than to B, the agent can decide that there is a higher payoff in executing the sequence B, A, C, D, even though A had the first priority.

6 Interaction between agents

Our agents interact by implicit and explicit communication. Implicit communication entails the observation of another agent's changes to the world. Implicit communication is possible in our system because the

[2] When a new service appears in the system it registers with a yellow page. When a service disappears, we use a lazy method to detect it. When an agent is sent to the location of a service that is out of business, the agent comes back to report this finding to the yellow page.

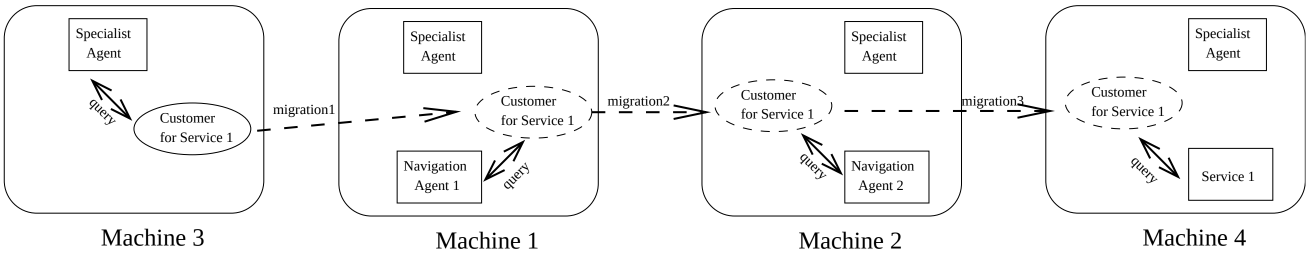
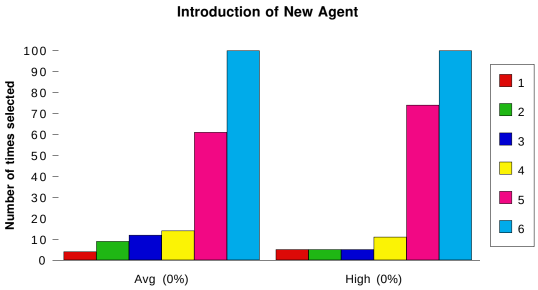


Figure 2: An example of navigation. Each machine has a number of fixed agents (denoted by rectangular blocks.) The *specialist agents* know about the location of one or more *navigation agents*. There are two navigation agents shown here: one on machine 1 and one on machine 2. The navigation agent on 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.



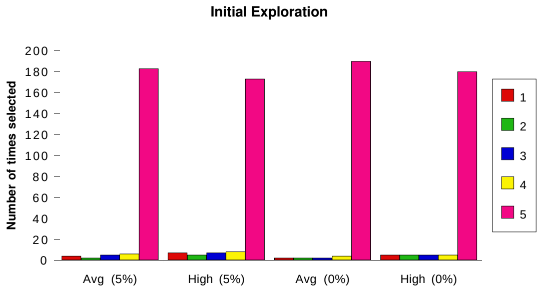


Figure 3: Selection of the best service. The services are listed in an increasing order of "goodness" and they are numbered 1-6 on the right hand side of the diagram. The left graph shows the service selection numbers when a new and better service (Agent 6) is added. The right graph shows the effects of randomization on the service selection numbers.

agents can use their sensors to observe changes in the environment as described in sections 4. Explicit communication is the direct exchange of information between two agents. In our system, two agents residing on the same or different machines can exchange messages and can open a dedicated connection for direct data transfer with the agent commands `agent_send`, `agent_receive`, and `agent_meet`. Agents need to know the current network location of the recipient as well as the unique symbolic name that the recipient has chosen for itself (the servers maintain a list of the agents that are executing on the current machine and keep track of their activity). An agent discovers another agent by using a hardwired name, consulting a specific Agent Tcl server, or consulting a virtual yellow page.

7 Information-Gathering Agents

We have used transportable agents for distributed information access. In distributed information gathering, a distributed collection of corpora is searched based on a query and the results extracted from each site are fused in a coherent picture.

We have built information-gathering agents that interface with the Smart information retrieval system[3]. Our data is a distributed collection of Smart repositories running the Smart system. Each collection consists of computer science technical reports. For a given

[3]The Smart system is a successful statistical information-retrieval system [Sal91] that uses the vector-space model to measure the textual similarity between documents. The idea of the vector-space model is that each word that occurs in a collection defines an axis in the space of all words in the collection. A document is represented as a weighted vector in this space. The premise of this system is that documents that use the same words map to neighboring points and that statistics capture content similarity.

query, an information agent visits a sequence of sites; at each site, it interacts with the local Smart agent to search the local collection. The results retrieved are brought home, or used as relevance feedback to refine the query.

In our experiment, the agent extracts a list of sites (DEC stations, PCs, and SGIs) that run Smart servers by consulting a virtual yellow page. The agent routes itself through the sites using the reactive-planning techniques outlined in Section 5, visiting each place exactly once. The query is run on the local server and a ranked list of documents is returned to the agent. Some simple error-detection and recovery mechanisms are incorporated into this system. If the plan of the agent takes it to a crashed or non-existent site, the error-recovery wrapper around the jump command enables the plan to continue. If the current site is down or on a low bandwidth connection, the agent greedily attempts to go to the next site. In our current implementation, if the Smart server crashes, the agent times out while waiting for the answer and continues the task at the next site. If the site crashes while the agent is there, the agent dies. A sample session from running this information-retrieval agent is shown in Figure 4.

We have extended this experiment using the mobile computer-support functions described in [GKNRC95] as follows. We started the information gathering agent on a laptop computer called Bond, and the agent immediately jumped off the laptop to interact with Smart agents throughout the network. Before the agent could return, we disconnected Bond, carried it to another lab, connected it to a different subnet, and reconfigured it with a new IP address. Meanwhile, the Information Gathering agent had finished its task and had attempted to jump back to Bond. The jump failed so this agent waited on a "dock" computer associated with Bond. When Bond reconnected to the network, it contated the dock, which then forwarded the Information Gathering Agent on to Bond.

8 Summary

We describe a system that implements autonomous software agents and illustrate an application of agents to distributed information gathering. We argue that autonomous agents require mobility and independent decision making. Mobility is an important attribute for dealing with an increasingly networked world. Independent decision making is critical for a mobile agent to adapt to a dynamic environment, especially when far from "home." We implement mobility with transportable programs (agents). As they travel, these agents sense the current network and software conditions and adapt their behavior to the sensed values. Our agents can be viewed as virtual robots that are equipped with virtual sensors and effectors and are capable of maintaining internal state, registering external state, and interacting with their environment.

References

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

[Bro86] R. Brooks, A robust layered control system for a mobile robot, in *IEEE Journal of Robotics and Automation*, 1986.

[DJR93] B. Donald, J. Jennings, and D. Rus, Information Invariants for Cooperating Autonomous Mobile Robots, in *Proceedings of the International Symposium on Robotics Research*, 1993.

[DJR94] B. Donald, J. Jennings, and D. Rus. Analyzing Teams of Cooperating Mobile Robots. In *Proceedings of the International Conference on Robotics and Automation*, San Diego, 1994.

[EW94] O. Etzioni and D. Weld, A softbot-based interface to the Internet, in *Communications of the ACM*, 37(7):72–76, 1994.

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

[GK94] M. Genesereth and S. Ketchpel, Software agents, in *Communications of the ACM*, 37(7):48–53, 1994.

[Gra95] R. Gray, Agent Tcl, in Proceedings of the CIKM Workshop on Intelligent Agents, Baltimore, MD, 1995.

[Gra96] R. Gray, Agent Tcl: A transportable agent system, in Proceedings of the Fourth Annual Tcl/Tk Workshop, Monterey, Ca, 1996.

[GKNRC95] R. Gray, D. Kotz, S. Nog, D. Rus, and G. Cybenko, Mobile Agents for Mobile Computing, Technical Report PCS-TR96-285, Department of Computer Science, Dartmouth College, 1996.

[JRS95] D. Johansen, R. van Renesse, and F. Schneider, Operating system support for mobile agents, in *Proceedings of the 5th IEEE Workshop on Hot Topics in Operating Systems*, 1995.

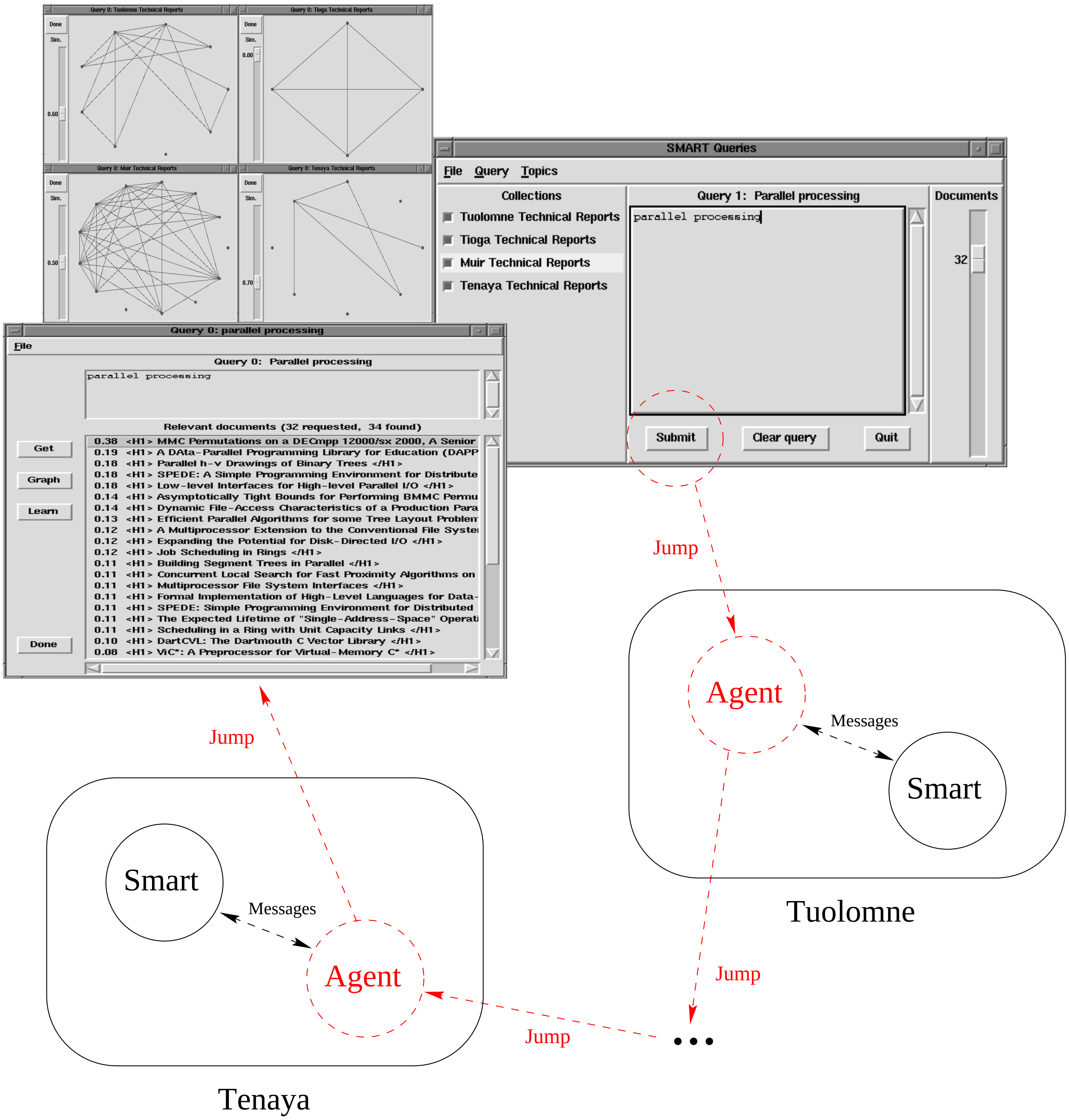


Figure 4: A sample session for the information-retrieval agent. The query screen is shown in the upper right corner of the figure. The agent follows the path described with dotted arrows from the home site to a first document collection on Tuolomne, to a second collection on Tioga, to a third collection on Muir, and finally, to the last collection on Tenaya. The agent returns to the home site and displays the results as (1) a ranked list of titles and (2) four graphs that show the inter-document similarities. The nodes in these graphs represent documents and the edges show similarity connections. The user may click on a node to view the text of the document

[KC86] R. Kahn and V. Cerf, *The World of Knowbots*, report to the Corporation for National Research Initiative, Arlington, VA, 1988.

[KSC94] H. Kautz, B. Selman, and M. Coen, Bottom-up design of software agents, in *Communications of the ACM*, 37(7):143–145, 1994.

[KK94] K. Kotay and D. Kotz, Transportable agents, in *Workshop on Intelligent Information Agents*, December 1994.

[KGR96] D. Kotz, R. Gray, and D. Rus, Transportable Agents Support Worldwide Applications, in Proceedings of SIGOPS96, 1996.

[Mae94] P. Maes, Agents that reduce work and information overload, in *Communications of the ACM*, 37(7):31–40, 1994.

[MCF94] T. Mitchell, R. Caruana, D. Freitag, J. McDermott, and D. Zabowski, Experience with a learning personal assistant, in *Communications of the ACM*, 37(7):81–91, 1994.

[NCK96] S. Nog, S. Chawala, and D. Kotz, An RPC mechanism for transportable agents, Technical Report PCS-TR96-280, Department of Computer Science, Dartmouth College, 1996.

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

[RS93] D. Rus and D. Subramanian, Multi-media RISSC Informatics: Retrieving Information with Simple Structural Components, in *Proceedings of the ACM Conference on Information and Knowledge Management*, Nov. 1993.

[RS96] D. Rus and D. Subramanian, Customizing Multimedia Information Access, *ACM Computing Surveys*, vol. 7, no. 4, 1995.

[RS96] D. Rus and D. Subramanian, Customizing Information Access, *ACM Transactions on Information Systems*, January 1997.

[Sal91] G. Salton. The Smart document retrieval project. In *Proceedings of the Fourteenth Annual International ACM/SIGIR Conference on Research and Development in Information Retrieval*, pages 356-358, 1991.

[SG90] J. Stamos and D. Gifford, Remote execution, in *ACM Transactions on Programming Languages and Systems*, 12(4):537–565, October 1990.

[TLKC95] B. Tomsen, L. Leth, F. Knabe, and P-Y. Chevalier, Mobile agents, ECRC external report, European Computer-Industry Research Center, 1995.

[Whi94] J. E. White, Telescript technology: The foundation for the electronic marketplace, General Magic White Paper, General Magic, Inc., 1994.