Software Agents for Information Retrieval

Ellen M. Voorhees
Siemens Corporate Research, Inc.
755 College Road East
Princeton, NJ 08540
ellen@scr.siemens.com

Abstract

The advent of large wide-area networks connecting many diverse repositories of data creates the challenge of finding particular data of interest in an easy and timely manner. This paper describes the initial prototype of a system that addresses this problem by using a set of autonomous, customizable software agents. Central to the system design are *scripts* — arbitrary, parameterized program segments written in a language that contains primitives for interacting with the data environment and tagged with keywords describing their purposes — that define the functionality of an agent. Agents can query the keywords associated with other agents' scripts and can import scripts from one another, thereby providing a mechanism for agents to learn from one another.

Introduction

Large, wide-area, high-bandwidth networks such as the Internet create a number of opportunities and challenges for effective retrieval of information. Such networks allow individuals to gain access to huge amounts of data of a wide variety of types. However, without effective information management tools most of this data is worthless since users are unable to find the data of interest to them — assuming they know of its availability in the first place. We have developed a prototype system that uses a collection of software agents as information retrieval tools for such an environment.

This prototype, the Information Retrieval Agent (IRA) system, focuses on the problem of retrieving texts from a set of document collections. It uses ideas — and code — developed for an earlier application in which agents control the flow of product through a factory [Wolfson *et al.*, 1989; Voorhees, 1991]. In the IRA application, agents are routed to a series of document collections[1], much as knowbots [Kahn and Cerf, 1988] and KNO's [Tsichritzis *et al.*, 1987] were envisioned as moving among resources to retrieve data. Unlike the product routing environment, however, an IRA models an individual user and adapts to that user's interests and preferences, and, eventually, will be able it to anticipate the user's queries, to complete missing information, and to retrieve relevant collateral sources. We believe this adaptability is crucial to forming an effective solution to the network retrieval problem, just as Etzioni and Segal view adaptability as a fundamental capability of softbots [Etzioni and Segal, 1992].

[1]In the current version of the prototype, explicit, physical movement of the agent to the collection is unnecessary and therefore not implemented. Once the system is deployed in a realistically large network, explicit movement will be necessary and accommodated.

The remainder of this paper describes the current IRA prototype in detail. In the conclusion we suggest areas to address so the system may evolve into a more complete information management system.

System Description

The IRA system has two types of agents, those that represent a corpus (*corpusbots*) and those that represent a user (*userbots*). A corpus is any collection of documents a provider is willing to allow others to search, such as back issues of the *Wall Street Journal*, *Grolier's Encyclopedia*, or the collected works of Edith Wharton. A corpusbot is the system's model of that particular corpus; it is the repository for all corpus-dependent parameter values and controls all access to its corpus. A corpusbot has both attributes and descriptors. Attributes describe the behavior of a corpusbot. For example, attributes can be used to differentiate among various kinds of access control strategies (free access to all, subscriber-only, etc.). Descriptors are topic designators that provide a high level summary of the contents of the corresponding corpus.

Similarly, a userbot is the system's model of a particular user; the userbot stores the user-dependent parameter values and is the user's intermediary in the system. User-dependent parameters include such items as X resources needed to customize the X window user interface to the userbot, and whether this user prefers recall- or precision-oriented searches (i.e., prefers to see everything of interest and therefore tolerates a large amount of irrelevant items, or is willing to risk missing some items of interest to avoid most irrelevant items). A userbot also maintains lists of the names of other userbots and corpusbots known to its user. In addition, a userbot contains a list of specialties. Like descriptors, specialties are topic designators, but in this case, the topics are presumed to be areas in which its user has interest and expertise. Specialties are arbitrary strings self-selected by the user, e.g., "information retrieval", "UNIX system kernel hacks", "hang gliding".

An agent is implemented as a long-lived process that reacts to typed messages sent to it by some agent (including possibly itself). Messages are processed in the order in which they are received and may cause the agent to perform some action. When there are no messages to process, the agent simply waits for another message to be received. An agent may send a message to any agent whose name it knows; it may also send a message to either the entire set of corpusbots or the entire set of userbots.

The ability to send a message to all corpusbots greatly facilitates resource discovery — a userbot need simply announce its request on a given topic and see which corpusbots respond. However, sending a single message to all corpusbots in a large, wide-area network is clearly infeasible. The IRA system can scale up to such a network by interpreting "all" as "all the agents in the local domain". The ISIS distributed toolkit [Isis Distributed Systems, Inc., 1992], which we use as our communication medium, provides support for domain-dependent communication. Explicit movement among domains as described in [Wolfson *et al.*, 1989] will allow userbots to find resources not within their immediate vicinity of the network.

Userbots

Userbots are the primary component of the IRA prototype system. Userbots send and receive messages among themselves and also send messages to corpusbots. A userbot responds to some basic messages such as requests for the contents of its specialty list or names of the corpora and users it knows. However, the bulk of a userbot's functionality is defined by the *scripts* it knows. Scripts are arbitrary, parameterized subprograms written in a language that contains primitives for interacting in the IRA environment, standard flow control and data manipulation statements, and the ability to execute a command at the operating system level (i.e., a function akin to C's "system" call). Scripts have keywords assigned to them that describe the overall objective of the script.

Figure 1 gives a (stylized) example of a script. This script's intended purpose is to search databases that are known to contain sports information for sports stories; it might thus be tagged with the keyword "sports". The function calls in boldface are IRA environment primitives. Some primitives send messages to agents and make the replies available in the data space of the script. For example, **select_corpora_desc**(x) sends a message to the userbot itself asking for the list of known corpora that have the descriptor x, and

```
q = "baseball football hockey basketball lacrosse";
corpora = select_corpora_desc("sports");
if (size(corpora) == 0)
        announce_failure("No sports databases.");

some = 0;
for each name_i in corpora do {
        results = query(name_i,q,15);
        if (size(results) > 0) {
                display_results(results);
                some++;
        }
}
if (some == 0)
        announce_failure("No sports news today.");
```

Figure 1: A script designed to retrieve sports articles.

query(*db, query-text, num-wanted*) submits a query to *db's* corpusbot. Other primitives provide access to the data thus returned (e.g., **size**) or handle I/O with the user interface program (e.g., **display_results** and **announce_failure**).

A separate user-interface program allows a user and his or her userbot to interact with one another. When invoked, the user interface program displays icons representing the set of corpora and users that are known to the userbot, and the list of scripts the userbot knows. Clicking on an icon causes the information associated with that particular entity to be presented. For example, the information associated with a corpus includes its attributes, its descriptors, and a history of its retrieval effectiveness (measured in terms of recall and precision). Selecting a script from the lists of scripts invokes that script. Once a script is invoked, the userbot interprets the associated program code and creates a complete execution trace (including parameter values; actions performed; results of individual actions, if any; errors encountered; etc.). This trace is currently used to show the path the userbot takes through the data space on the user's display. In the future it can be used as the episodic memory from which the userbot can refine its model of the user and thereby improve its effectiveness [Stanfill and Waltz, 1986; Maes and Kozierok, 1993].

A userbot is initialized with a default set of scripts. This set includes scripts that allow the user to send and receive messages for all the userbots' basic message types: search for databases that have particular attributes or descriptors, search for users that have particular specialties, and submit a query to a particular database. Non-null results of the first two scripts are included in the userbot's list of known corpora or known users. Default scripts also exist for searching the keywords associated with another (known) user's scripts, importing another user's script into one's own userbot, and incorporating newly-authored scripts into the userbot.

The ability to incorporate new scripts into a userbot is a powerful method through which users can extend and customize the functionality of their userbots. Series of search commands that are frequently repeated (such as an early-morning search for the previous day's sports scores) can be codified in a script; the entire sequence can then be invoked in a single command. Furthermore, since the scripts may contain other arbitrary program segments, the same script can manipulate the retrieved data however the user desires.

Importing scripts from other users makes customization even easier. Suppose, for example, that Alan is looking for documents about a new tax law, but doesn't know where to look. He knows, however that Barbara is a tax-specialist (or, he queries the local domain to see if there are any users with "taxes" or "tax attorney" as a specialty), so he searches Barbara's scripts for one tagged with "taxes" as a keyword. Assuming he finds one, he can import it and use it to submit his tax question, thereby immediately benefiting from Barbara's experience with tax databases. Of course, users may not desire others to be able to see certain scripts, and the prototype allows users to mark scripts as private. Still, there are a vast number of other privacy and security concerns in a system such as this that the prototype does not address.

Since all of a user's interaction with the IRA system occurs through his or her userbot, the userbot has the potential to become an expert on its user. Long-standing interests of the user can be used to form interest profiles that can be refined with relevance data using filtering techniques [Belkin and Croft, 1992]. Further, the userbot can build user-dependent models of the utility of particular corpora (e.g., this user almost always finds articles from the *Wall Street Journal* relevant, but seldom likes articles from the *Federal Register*). The effectiveness data currently stored in the known corpora list is a rudimentary form of such a model.

Corpusbots

Corpusbots are fairly pedestrian in the current prototype. A corpusbot sends no messages and responds to exactly four message types:

- a request for its attributes receives the attribute list in reply;
- a request for its descriptors receives the descriptor list in reply;
- a message containing a natural language query and a number of documents to be retrieved receives a list of that many [document id, document title, similarity value] triples in reply; and
- a message containing a list of document ids receives the texts of those documents in reply.

Both retrieving documents in response to a query and fetching text given a document id are implemented using the SMART information retrieval system [Salton and McGill, 1983].

Just as a userbot may become an expert on its user, a corpusbot is the single access point to its corpus, and thus it has the potential to become an expert on the corpus. The corpusbot is the logical repository for relevance data across all users of the corpus. It can then use this data to improve its indexing of the corpus using, for example, the technique suggested by Fuhr and Buckley [Fuhr and Buckley, 1991].

Conclusion

We have developed a prototype system that uses software agents to retrieve documents from disparate databases. As a prototype, the system has some limitations in both implementation and design. The current implementation does not permit concurrent execution of scripts in a single userbot, nor does it allow one script to call another. As a consequence, userbots remain idle unless a user is actively engaged with the system. More fundamentally, these agents do not yet have the initiative and adaptability required for a truly adequate solution to the network retrieval problem.

Nevertheless, the IRA system has several interesting features. The direct transfer of ideas and code from the product routing application to the network retrieval application demonstrates that agents as user-written programs are applicable in multiple problem domains. The prototype as it exists today can automate routine retrieval tasks, and can accommodate some measure of resource discovery by capitalizing on others' experience. At the very least, the use of one script by many different individuals leverages the cost of writing the script. Finally, the prototype provides a test-bed for continued research into developing more sophisticated agents.

One of the more important ways we hope to increase the sophistication of the agents is in their ability to model their users and corpora. As suggested above, relevance information from the users should be able to be exploited so that the agents' models evolve over time in such a way as overall effectiveness is improved.

References

Nicholas J. Belkin and W. Bruce Croft. Information filtering and information retrieval: Two sides of the same coin? *Communications of the ACM*, 35(12):29–38, December 1992.

Oren Etzioni and Richard Segal. Softbots as testbeds for machine learning. In *Working Notes of the 1992 Spring Symposium on Knowledge Assimilation*, pages 43–50. American Association for Artificial Intelligence, March 1992.

Norbert Fuhr and Chris Buckley. A probabilistic learning approach for document indexing. *ACM*

Transactions on Information Systems, 9(3):223–248, July 1991.

Isis Distributed Systems, Inc. *Isis Distributed Toolkit Version 3.0 User Guide and Reference Manual.* Ithaca, NY, 1992.

Robert E. Kahn and Vinton G. Cerf. The Digital Library Project Volume 1: The world of knowbots (DRAFT). Corporation for National Research Initiatives, March 1988.

Pattie Maes and Robyn Kozierok. Learning interface agents. In *Proceedings of the Eleventh National Conference on AI (AAAI-93).* American Association for Artificial Intelligence, July 1993.

G. Salton and M. J. McGill. *Introduction to Modern Information Retrieval.* McGraw-Hill, New York, 1983.

Craig Stanfill and David Waltz. Toward memory-based reasoning. *Communications of the ACM*, 29(12):1213–1228, December 1986.

D. Tsichritzis, E. Fiume, S. Gibbs, and O. Nierstrasz. KNOs: KNowledge acquisition, dissemination, and manipulation Objects. *ACM Transactions on Office Information Systems*, 5(1):96–112, 1987.

Ellen M. Voorhees. Using computerized routers to control product flow. In *Proceedings of the 24th Annual Hawaii International Conference on System Sciences (HICSS-24)*, volume 2, pages 275–282, January 1991.

C. Daniel Wolfson, Ellen M. Voorhees, and Maura M. Flatley. Intelligent routers. In *Proceedings of the Ninth International Conference on Distributed Computing Systems*, pages 371–376. IEEE, June 1989.