How to Personalize the Web

Rob Barrett Paul P. Maglio Daniel C. Kellem
IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120
`barrett,pmaglio,kellem@almaden.ibm.com`

ABSTRACT

Agents can personalize otherwise impersonal computational systems. The World Wide Web presents the same appearance to every user regardless of that user's past activity. Web Browser Intelligence (WBI, pronounced "WEB-ee") is an implemented system that organizes agents on a user's workstation to observe user actions, proactively offer assistance, modify web documents, and perform new functions. WBI can annotate hyperlinks with network speed information, record pages viewed for later access, and provide shortcut links for common paths. In this way, WBI personalizes a user's web experience by joining personal information with global information to effectively tailor what the user sees.

Keywords

Agents, World wide web, User models.

Download

WBI may be downloaded from http://www.raleigh.ibm.com/wbi/wbisoft.htm

INTRODUCTION

One goal of networked computing is to provide all users access to the world's vast information resources. This trend toward a single, global database clearly shows its potential in the World Wide Web. Tens of thousands of institutions, hundreds of millions of pages of information, simple means for authoring, and simple means of access have combined to make the web a place to find information on just about any topic. One way to describe the web is to say that every information resource is **equally proximate** to a user. Simply specify a universal resource locator (URL) and the hosting machine will deliver the page, picture, sound, or video to your workstation within seconds. This method of information access is fundamentally changing the way people handle and think about information [12].

URLs make all information equally available to a user. Although the prospect of having 100 million pages close at hand is staggering, hypertext simplifies access to this huge information repository. Web authors arrange information into directed graphs that link related pages together. In this way, hypertext reduces the user's decision space from a choice among millions of URLs to a choice among tens of hyperlinks on a particular page. Thus, hypertext transforms a large number of pages that are logically proximate into an accessibility space in which some pages are practically closer than others. Put differently, the web combines universal proximity (URLs) with tractable accessibility (hypertext). Given this view, it seems that the accessibility of information is completely impersonal. Every user sees the same web connected together in the same way. Web authors build the structure of the web according to their interests, concerns, and tastes, and then all users rely on that same structure.

To overcome the impersonal feel of the web, browser software generally provides "Hot Lists" or "Bookmarks" which allow users to record often-used URLs for quick access. These lists of URLs begin to make the web more personal, pulling certain web pages closer to individual users, and thus enabling easier access. They add a personal structure to the impersonal structure of the web. Of course, some users go further and author their own home pages which can also provide launching points for web access. In addition, browsers can generally be configured to automatically start up with a particular page loaded. Creating hot lists, home pages, and setting start pages are steps individual users can take toward personalizing the web.

The impersonal organization of information on the web has some negative consequences for users. In a recent survey, Pitkow and Kehoe found the main problems people report when using the web are (a) slow network or connection speeds; (b) not being able to find particular pages, even after they have been found before; (c) not being able to manage or organize retrieved information; and (d) not being able to visualize where they have been [16]. The first problem (beyond technical issues of modem speed and so on) results from the perception that all information is equally proximate. Because network delays routinely slow user access to particular information sources, the perceived logical proximity of information does not correspond to actual time needed to obtain information. The second and third problems result from users having to manually manage their personal information. The fourth problem results from the impersonal organization of the web . We believe all these problems can be solved by providing automatic mechanisms that tailor the web for individual users.

In this paper, we describe a fully implemented system, Web Browser Intelligence (WBI, pronounced "WEB-ee"), that **automatically** personalizes the web using agent technology [9]. WBI provides an architecture which taps into the communication stream between a user's web browser and the web itself. Small programs, or agents, attach themselves to this stream, observe the data flowing along the stream, and alter the data as it flows past. These agents can learn about the user, influence what the user sees by marking-up pages before passing them on, and provide entirely new functions to the user through the web browser. In what follows, we describe the WBI architecture, the style of interaction between the user and the web that we advocate, the specific functionality we have provided, and the future of WBI.

WBI ARCHITECTURE

The fundamental communication mechanism of the web is the hyper-text transfer protocol (HTTP), although others such as gopher and file transfer protocol (FTP) are also used and can be treated analogously. HTTP is a simple, stateless, request-response system [2]. A browser connects to a server, sends a retrieval request, the server sends the requested document and then closes the connection. HTTP also allows a proxy to mediate this transaction: the browser sends the request to the proxy, which retrieves the document from the appropriate server and then returns the document to the browser. The proxy mechanism is often used to provide a one-way firewall for intranet security. WBI is a proxy that intercepts the HTTP stream for observation and alteration. Every web transaction flows through WBI as a request goes from the browser to the web and the response returns back to the browser (see Figure 1).

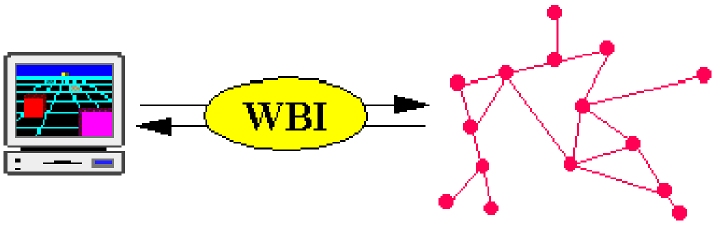

Figure 1. WBI stands between a user's web browser and the web, monitoring, editing, and generating documents.

Four Kinds of Agents

WBI is composed of three types of agents which interact with this request-response stream (monitor agents, editor agents, generator agents), and one type of agent which acts independently (autonomous agents). A monitor receives a copy of the entire request-response transaction but cannot alter it. Monitors track user actions to provide information for other agents. For example, we have a page content monitor that records all of the text contained in the web pages that a user has viewed. The resulting content history can then be used to access previously-viewed pages via keyword searching. Any number of monitors may observe any particular transaction.

An editor agent intercepts the communication stream, receiving information and then delivering a modified version of it. This edited data stream can be created from the incoming stream and other available information, such as a user's past history, system status, or any information obtained from the web. Editors can connect to either the request part of the stream or to the response part. Request editors can transform a document request from the browser into another request. One simple application of a request editor is to fetch documents that WBI knows have been moved to a different URL. Response editors can modify the actual content of documents that users see. For example, response editors are used to add extra buttons or additional links to a web page, or to change colors and backgrounds. Alternatively, editors can choose to simply pass their input to their output, effecting no change in the stream. Any number of editor agents may alter a transaction.

A generator is an agent that converts a request into a response. Every transaction activates exactly one generator. The default generator is simply a web communication program that passes the request on to the appropriate web server, retrieves the response and passes it back to the browser (i.e., simply performing the job of an HTTP proxy). Other generators are used to provide new WBI functions, intercepting requests from the browser and generating documents for the user to see in response. For example, a generator could provide a web-based way to view the state of the printer queues on a user's workstation. When the generator is activated, it produces an HTML document which describes the printer queues. Likewise, a generator can handle requests with special communication protocols, such as communicating with a firewall.

Finally, an autonomous agent is executed based on a trigger independent of the communication stream, such as a time interval. An autonomous agent simply performs its task and then terminates. For example, an autonomous agent might perform housekeeping actions, digest recorded transactions to develop user models, or explore the web for new information.

How Agents are Triggered

WBI's central loop dynamically constructs a network composed of monitors, editors, and a generator to handle each request and response. At start up, each agent registers itself with the arbitrator along with its trigger rules for activation. Agents can be triggered when particular servers or domains are accessed, when particular document types are received, or at particular times or intervals. Agents also register ordering and grouping information so that (a) editors can control the order in which they receive the communication stream (to work cooperatively in modifying the stream), and (b) monitors can control whether they see the response as it comes from the generator, as the user sees it, or after a particular editor has modified it. Generators register the types of requests that they handle. Often generators register new URLs that do not exist on the web but that can be addressed as normal web documents. For example, a generator that displays a workstation's printer queues might trigger on a request to http://wbi/printq. Thus, every request is specially handled by the arbitrator, which routes it through the appropriate sequence of agents.

How Agents Work Together

As an example of how agents can work together to achieve a new function, consider how a collection of agents can point a user toward previously undiscovered but potentially interesting resources on the web (see Figure 2). Roughly, WBI monitors the web activity of a particular user, classifies accessed web pages according to subject, searches the web for additional pages related to subjects the user has browsed, and adds links to subsequently accessed pages that suggest additional pages of interest. More precisely, a monitor is triggered whenever a document containing text is retrieved. This agent records the text of these documents. An autonomous agent is triggered on a time interval, periodically digesting the text the user has viewed, grouping the pages into clusters, and extracting keywords that describe the clusters. A second autonomous agent, triggered by the completion of the first autonomous agent, contacts global web search services with the lists of keywords to discover new documents that might be interesting to the user. An editor agent watches each document the user browses, looking for a URL known to be in one of the subject areas for which new documents have been found. When such a URL is observed, an alert is added to that web page notifying the user that related information is available. Finally, when the user clicks on the notification, he or she is directed to a generator that displays the list of related pages to be examined. Thus, fairly simple agents can work together to produce new functions to intelligently and automatically personalize the web.

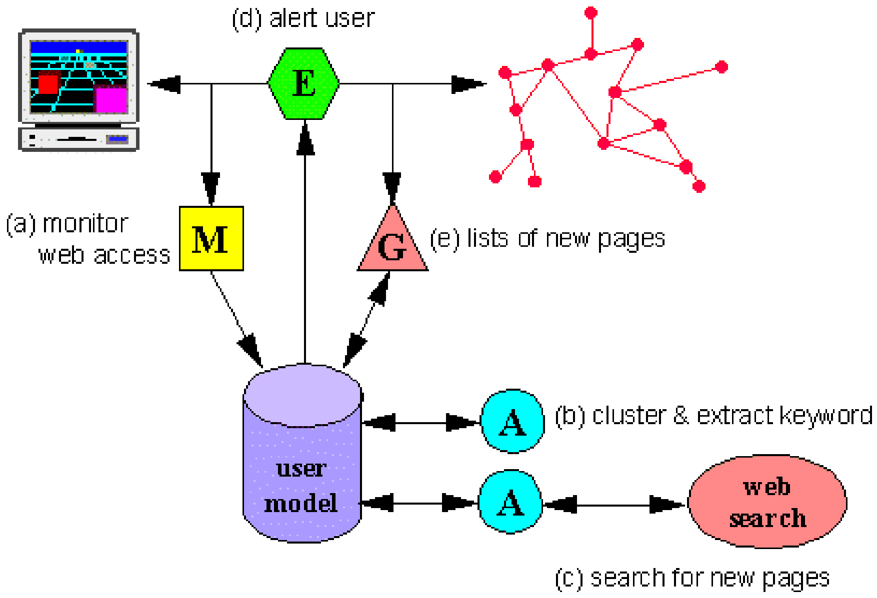

Figure 2. A collection of agents that (a) monitors web access, (b) clusters retrieved pages and extracts keywords to identify clusters, (c) searches for new pages containing the keywords, (d) edits retrieved pages that are part of known clusters to point to alert user of related pages, and (e) generates pages containing lists of related URLs.

Performance Requirements

WBI has been implemented as a proxy server that can run either on the user's client workstation or on a server that many users access. The core of WBI (a) receives HTTP, gopher, and FTP requests, (b) constructs a network containing monitors, editors, and a generator based on trigger conditions to handle the request, (c) feeds the communication stream to each agent, and (d) returns the response to the browser. WBI also launches autonomous agents when their trigger conditions are satisfied.

Fast performance is a key requirement. Because WBI stands in the middle of every web transaction, if it noticeably slows down communications, people will not use it. Typical web transactions take between 0.5 and 20 seconds. We designed WBI and its associated agents to add no more than 1 second of overhead to these transactions. To perform at this level, WBI was built in a highly multi-threaded fashion so that no byte that ought to be delivered to the browser gets stuck somewhere in the agent network. Generators and editors control what is delivered to the browser so they are run at high priority. Monitors are run at a low priority, as they do not affect what a user sees.

Because of performance considerations editor agents must always pass data along the stream as quickly as possible. This implies that editors should not depend on the contents of whole documents to make their modifications, but should depend only on small, localized sections of documents. For example, an editor that annotates the top of a web page only if the page contains a certain word or phrase or link might search the entire contents of page before it passes the page along. Given large web pages that contain hundreds of kilobytes of text, the delay incurred by such an editor could be substantial, lasting tens of seconds. One way to put information at the top of a page that depends on information later in the page is to use image embedding. An editor can insert an embedded image that points to a generator that produces the appropriate image. This trick enables the editor to pass the stream along while processing the data, effectively delaying the decision about whether to annotate until enough data has been seen. Of course, this only works for graphical browsers that incrementally display pages as they load.

Related Work

The Open Software Foundation's OreO (now called Strand) is a proxy server shell that can be used to modify the HTTP stream between a client and a web server [3]. In this way, OreO is similar to WBI. However, our implementations differ substantially. First, OreO has static connectivity between agents configured at start up, whereas WBI dynamically connects agents based on data contained in request and response. Dynamic configuration allows agents to respond to certain requests and ignore others. WBI's rule system arbitrates among possible monitors, editors and generators, dynamically constructing a chain of agents to pass the stream through. Whereas in OreO, editor chains must be manually constructed by literally plugging proxies together, WBI configures editor chains automatically.

A second difference between WBI and OreO is that OreO does not provide any generator function. OreO expects an independent HTTP proxy server at the end of the chain that will actually retrieve a web document. Thus, OreO cannot serve dynamic documents itself, but must rely on a separate server. As mentioned, WBI incorporates generator agents which can produce documents in response to HTTP requests. The example applications we have built often required generator agents for producing search forms and results, dynamic images, configuration pages, and so on. Thus, generators are integral to agent-user interaction.

Third, OreO provides no special support for monitor functions. Although monitors can be written as editors that simply pass their data along while storing what they need, many such monitors chained along the HTTP stream will inevitably slow the movement data along the stream. Because WBI distinguishes monitors from editors, monitors can run at a lower process priority, maintaining the throughput from web to user.

Finally, an OreO-based proxy acts as a single editor for transforming HTTP requests and responses. To create a set of agents using OreO, a set of independent proxies must be created. Using WBI, a single proxy which contains a series of separate transformations can be used instead. The processor overhead of creating a series of socket connections to chain OreO proxies can be substantial.

Before describing details of our agents, we next discuss several human-computer interaction guidelines that have influenced the design of WBI.

USERS SEE THE WEB, NOT WBI

WBI can be used to dramatically transform what a user sees on the web. For example, all interaction can be based on explicit conversations with an agent rather than with the web directly (as in [3]). However, the set of agents we have built were designed to operate in a style we believe is best suited to assisting users in their **normal use of the web**. That is, rather than radically changing the web experience, our aim has been to add new functions without substantially altering a user's ordinary interaction [15]. Toward this end, we have followed five guidelines in developing WBI's user interface.

1. WBI must maintain the standard user interface of the web---the user communicates principally and directly with the web rather than with agents. This means that all of WBI's output and all user input must be done via HTTP and HTML. Users click on links or type in URLs. WBI's agents display their results on ordinary web pages.
2. Agents act both proactively and reactively. When they have nothing to say, agents must remain silent and hidden. When an opportunity for augmentation arises, however, proactive assistance and advice can be offered. A reactive response results from a direct request for an agent's assistance. For example, when a user requests http://wbi/printq to display workstation printer queues, the printer queue generator agent reacts by reporting on the queue state.
3. Agents cannot take control from the user. Agents can offer advice, but the user remains in charge. This reduces the problem of user-agent trust which can hamper agent systems.
4. Agent activity must be peripheral (not central) to the task at hand so the user can easily ignore errant agent activity.
5. Agents build user models which generalize, recognize, and classify user activity to predict future activity. For example, in recognizing that a user reads a particular web page every morning, one agent has built a model of the user that enables another to prefetch that page or to notify the user if he or she forgets to read that page one day.

These guidelines have also influenced the development of other agent systems in our research group, such as the COACH adaptive help system [18].

We now turn to details of WBI's agents.

SIMPLE AGENTS SOLVE COMMON PROBLEMS

We implemented a basic set of WBI agents to solve some of the more common problems users experience on the web. As mentioned, web users report that they have trouble re-finding pages that they have found before, and that they become frustrated when servers are either down or subject to network delays [16]. People also like to keep track of particular web pages that are likely to change often. The basic set of functions we implemented include a personal history, shortcut links, page watching, and web traffic lights. Similar functions have been built by others as separate, monolithic applications (such as [8, 14]). By contrast, WBI is configurable and extensible, providing a single, uniform mechanism for injecting automatic and intelligent functions into communications between user and web. Our set of basic agents comprise only the first examples we constructed using the WBI architecture.

We describe each of WBI's basic functions in turn, and then discuss more complicated functions.

Personal History

The personal history is recorded by a monitor agent that stores the sequence of URLs visited by the user along with the text of each URL. The text is stored in an inverted index to allow rapid retrieval based on keywords [17]. The user can access this personal history through keyword queries or path browsing. Keyword queries retrieve a list of pages the user has viewed sometime in the past that contain the given keywords. Thus the problem of re-finding a previously-viewed page is reduced to a query over several thousand relevant pages, rather than a query over hundreds of millions of pages on the web at large. Path browsing allows a user to view paths he or she has taken through a particular page. If the user knows only that some relevant page was seen shortly after going through the IBM home page, then by browsing only the paths taken through the IBM home page, he or she is likely to quickly find the sought-after document. In fact, people might actually conceptualize the web as motion along paths rather than as static URLs [13]. In any event, both keyword queries and path browsing are easily implemented with generators that access the personal history. Figure 3 shows an example of the personal history screen.

Figure 3. This screenshot shows part of WBI's personal history function. The form near the top of the page can be used to limit the displayed URLs to those that contain a specific word or phrase.

Shortcuts

The shortcut editor agent also relies on the personal history. Experience suggests that users often follow the same subsequence of URLs many times. For example, one of us routinely looks up documentation on a certain programming language by going to the language's home page, then to programming information, then to language information, and then to the reference manual. The shortcut editor observes this pattern and adds links to the first page in the sequence so that the user can skip the intermediate pages (see Figure 4). It does this by searching the sequence of URLs visited by the user for patterns of activity following an access of the current URL. If such a pattern is found, the URLs corresponding to the following accesses are added as links to the current URL. These shortcuts provide dynamic annotations to web pages based on a simple user model.

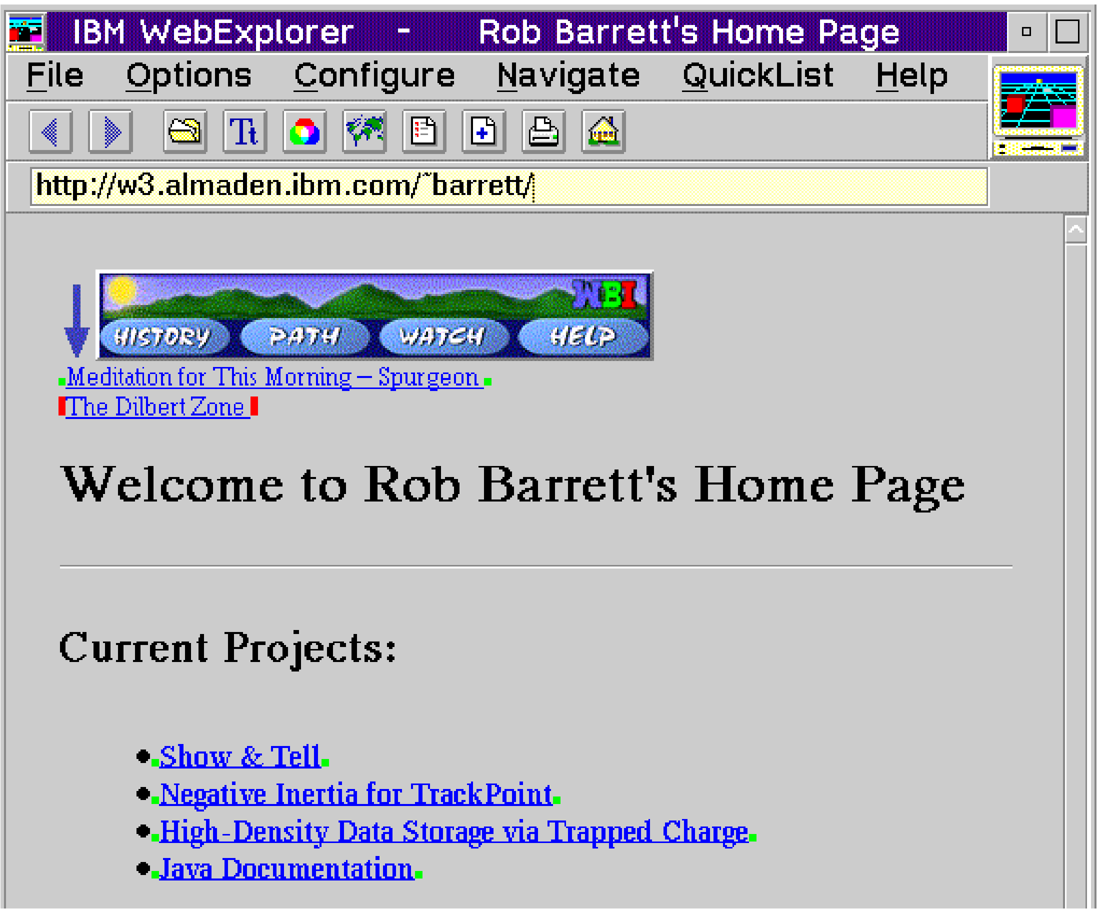

Figure 4. This screenshot shows the top of a web page on which one editor agent has added a toolbar for accessing WBI's main functions (History, Path, Watch, and Help), and another editor has added shortcut links. The hyperlinks under the arrow to the left of the toolbar are URLs that this user has often visited shortly after visiting the current URL, as sketched in the text.

Page Watching

The page watching agent notifies a user when certain pages change. Page watching can be used as a subscription to dynamic pages, for example, newsletters, meeting announcements, and comic strips. In WBI, page watching is implemented as an editor that adds a link to the top of every page. The user can follow this link to indicate that the current URL is to be watched. This editor also adds alerts to web pages whenever it needs to inform the user that a watched page has changed. The alert is a link that can be followed to the newly updated page. An autonomous agent is triggered on a time interval to check whether watched pages have changed.

Traffic Lights

One of our most popular editor/generator combinations is the **web traffic light**. This function annotates each HTML page with colored dots around each hyperlink to indicate the speed of the network link to that particular server. An editor agent adds small in-line images around each link on a web page. These images are served by a generator agent which maintains a database of network delays to web servers. Servers that are not in the database are pinged to determine the delay. Delay times in the database are rechecked periodically in an effort to balance information timeliness with network traffic. The traffic lights instantly inform the user about expected delays, enabling both efficient choice among equivalent links and mental preparation for ensuing delays. Because most browsers are multi-threaded, display of the actual page is not affected too much as network conditions are determined; rather, the dots appear beside the links as network speed information is gathered.

All these functions personalize the web for individual users. Frequently or previously visited pages are more easily accessible than the web at-large, and web performance (network speed) is determined in real-time and displayed in a useful way. The user's normal interaction with the web is maintained but with new, unobtrusive functionality.

COMPLEX AGENTS BUILD USER MODELS

Beyond the basic functions, we have also implemented agents that build complex user models to facilitate web browsing. For instance, one agent analyzes recently viewed pages to determine consistent trends in word usage, and then edits web pages to highlight links which seem to follow the pattern among pages the user is currently browsing (similar to [10]). Another set of agents finds undiscovered pages of interest from global web search services based on personal history clustering and keyword extraction, as described previously (and in Figure 2).

We are currently working on a set of agents that allow users to annotate and group web browsing activity into useful collections with minimal extra effort (cf. [5]). These collections capture some of the thought and understanding that went into a particular searching or browsing session [7]. They can then be referred to later so that the abstraction underlying **web history** can be raised from single pages to reasoned sessions [4]. We are also looking toward sharing histories among users so that expertise in a subject area can be transferred.

Finally, we have implemented agents that make a user's workstation desktop available through the web browser; these functions include launching applications, querying system state, managing files, and so on. Through networking, such agents can enable sharing work via the web.

HAS WBI SUCCEEDED?

One measure of WBI's success is the extent to which it affects people's use of the web. Data gathered by an email survey of more than 120 people who have used WBI suggests that WBI users have in fact changed their behavior. When directly asked, "Has WBI changed the way you use your browser's hotlist or bookmark feature?", more than half responded that it had. In particular, WBI users reported that they rarely use bookmarks, preferring instead to rely on (a) the personal history function to automatically save visited URLs and to facilitate searching through visited URLs, and (b) the watch function to return to URLs whose content has changed.

At first, we were surprised the survey also indicated that no single WBI function was most popular, and that most WBI user's clearly had a favorite function. Each individual web user seemed to find a single function to resonate with his or her standard way of using the web; for instance, whereas one person preferred to query the personal history using a keyword search, another preferred to scan the most recently visited sites on the personal history page. Yet the suggestion that individual users stick with a single method accords with data we recently collected by watching experienced web user search for specific information [11]. In that study, we found that individuals rely on standard routines or patterns of behavior. These behavior patterns are so strong that people not only plan and execute their searches according to their patterns, but they also recall their behavior as fitting their standard pattern even when it did not.

Further study will be required to understand how users are affected by the new functions WBI provides. The results of such studies will help us design the best functions. The remaining challenge is to unobtrusively encourage users who seem set in their ways to broaden their range of search strategies. The simple fact that WBI has been downloaded more than 25,000 times in six months suggests that people are actively looking for tools to enhance web use.

FUTURE DIRECTIONS: SCENARIOS & MODELS

We have been confronted with two basic challenges in creating WBI agents: scenario design and user modeling. Scenario design demands a human-agent-web interaction that is always convenient, informative, and usable. In our scenarios, we have tried not to change human-web interaction to facilitate human-agent interaction because the web is the primary entity that the user is working with. An alternative scheme would put users in direct contact with agents that travel the web or perform other actions on the users' direct instructions, such as electronic commerce scenarios. Of course, WBI can handle such scenarios using specialized generators that enable users to communicate with agents. In fact, we have found that making agents peripheral and unobtrusive can cause bottlenecks when agents need to tell the user something. For instance, when an autonomous agent has found a web page that it thinks the user wants to know about, how should that information conveyed? By interrupting the user's current activity with an announcement? By waiting until the user asks the agent? By subtly informing the user that it has something to say? By waiting until the user seems interested and available? The design of useful scenarios is a constant challenge.

User modeling will become increasingly important as our agents become more sophisticated. Modeling consists primarily of taking information from the communication stream and recognizing, generalizing, and classifying that information to produce useful results for other agents. This activity requires: (a) inferring what the user is thinking based on what the user is doing on the computer, and (b) determining what the computer should do based on what the user is (presumably) thinking. We have built a few simple examples of these transformations, such as the agent that derives clusters and keywords from the personal history and then finds documents containing those keywords. This agent tries to determine what is of interest to the user---under the assumption that keywords identify interesting documents [1]---and then translates those keywords into undiscovered documents using a search service. This is a relatively simple example of user modeling, but more sophisticated techniques will be required as user-interaction scenarios become more complex.

CONCLUSION

We have implemented a multi-agent system for assisting web browsing. WBI's architecture consists of small monitor, editor, generator, and autonomous agents that are hooked into the web communication stream through an arbitrator and a set of trigger conditions. We have implemented a number of agents that personalize the web for a user and that solve common web problems. Future agents will extend the ideas of personalizing the web experience, melding global and personal information, and enabling collaboration among web users.

As a final note, we believe the ideas behind WBI are not limited to the world-wide web but apply equally to any communication stream between a user and a computational system. Other candidates for study include workstation window environments, conventional information retrieval and database applications, online public-access catalogs, electronic mail, network news, command shells, word processors, and other applications.

REFERENCES

1. Belkin, N. J., and Croft, W. B. Retrieval techniques. *Annual Review of Information Science and Technology.* Elsevier, New York, 1987, pp. 109-145.
2. Berners-Lee, T., Fielding, R., and Frystyk, H. Hypertext transfer protocol: HTTP/1.0. Tech. Rep. RFC 1945, MIT, May, 1996.
3. Brooks, C., Mazer, M. S., Meeks, S., and Miller, J. Application-specific proxy servers as http stream transducers. In *Fourth International World Wide Web Conference* (1995).
4. Bush, V. As we may think. *Atlantic Monthly* (1945).
5. Card, S. K., Robertson, G. G., and York, W. The book and the web forager: An information workspace for the World-Wide Web. In *CHI 96 Conference Proceedings* (Vancouver, BC, 1996), Association for Computing Machinery, pp. 111-117.
6. Etzioni, O., and Weld, D. A softbot-based interface to the internet. *Communications of the ACM 37*, 7 (1994), 72-76.
7. Fertig, S., Freeman, E., and Gelertner, D. "Finding and reminding" reconsidered. *SIGCHI Bulletin 28*, 1 (1996), 66-69.
8. HitachiSoft. ZooWorks. Avaliable as http://www.zoosoft.com/zooworks.
9. IBM. Web Browser Intelligence - Agent Software. Available as http://www.raleigh.ibm.com/wbi/wbisoft.htm.
10. Lieberman, H. Letizia: An agent that assists web browsing. In *International Joint Conference on Artificial Intelligence* (1995), pp. 924-929.
11. Maglio, P. P., and Barrett, R. How to build modeling agents to support web searchers, 1996. Manuscript submitted for publication. IBM Almaden Research Center.
12. Marchionini, G. *Information Seeking in Electronic Environments.* Cambridge University Press, Cambridge, England, 1995.

13 Matlock, T., and Maglio, P. P. Apparent motion on the World Wide Web. In *Proceedings of the Eighteenth Annual Conference of the Cognitive Science Society* (San Diego, 1996), Lawrence Erlbaum, p. 810.

14 NetMind. The URL-minder. Avaliable as http://www.netmind.com/URL-minder/URL-minder.html.

15 Norman, D. A. How might people interact with agents. *Communications of the ACM 37*, 7 (1994), 68-71.

16 Pitkow, J. E., and Kehoe, C. M. Emerging trends in the WWW user population. *Communications of the ACM 39*, 6 (1996), 106-108.

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

18 Selker, T. COACH: a teaching agent that learns. *Communications of the ACM 37*, 7 (1994), 92-99.