

WBI: A Confederation of Agents that Personalize the Web

Rob Barrett and Paul P. Maglio and Daniel C. Kellem

IBM Almaden Research Center
650 Harry Road
San Jose, CA 95120

Abstract

Agents offer the possibility of personalizing otherwise impersonal computational systems. The World-Wide Web is an information collection that 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 provides a loosely confederated group of agents on a user's workstation capable of observing user actions, proactively offering assistance, modifying resulting web documents, and performing new functions. For example, WBI will annotate hyperlinks with network speed information, record pages viewed for later access, and provide shortcut links for common paths. WBI is an architecture in which small programs, or agents, connect to the information stream by registering their trigger conditions and then performing operations on the stream. This structure provides rich opportunities for personalizing the web experience by joining together personal and global information, as well as enabling collaboration among web users.

Introduction

One way to describe the World-Wide Web is to say that all information resources are equally *proximate* to all users. By arranging information into directed graphs that link related pages together, the hypertext organization of the web reduces a user's decision space from a choice among millions of information resources to a choice among tens of hyperlinks on a particular page. Thus, hypertext transforms a large number of pages that are logically close into an accessibility space in which some pages are practically closer than others. Put differently, the web combines universal proximity with tractable accessibility. Nevertheless, the accessibility of information is completely impersonal. Every

user sees the same web connected together the same way.

In this paper, we describe an implemented system, Web Browser Intelligence (WBI, pronounced "WEB-ee"), that *automatically* personalizes the web using agent technology (available at <http://www.raleigh.ibm.com/wbi/wbisoft.htm>). WBI provides an architecture which taps into the communication stream between a user's web browser and the web itself. Small programs, or agents, can 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, some performance considerations, and some of the specific functionality we have provided.

WBI Architecture

The web's hypertext transfer protocol (HTTP) is a simple, stateless, request-response system (Berners-Lee, Fielding, & Frystyk 1996). 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. 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.

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, a page content monitor 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.

An editor agent intercepts the communication stream, receiving information and then delivering a modified version of it. This edited stream can be created from the incoming data 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.

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. Other generators are used to provide new 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. Likewise, a special generator can be used to handle requests outside of a firewall or with specialized communication protocols.

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 arbitrates among possible agents to determine which receives the request from the browser, dynamically constructing a network composed of monitors, editors, and a generator to handle the request and response. 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.

As an example of how agents can work together to achieve a new function, consider how a confederation of agents can point a user toward previously undiscovered but potentially interesting resources on the web (see Figure 1). 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 that intelligently and automatically personalize the web.

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.

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

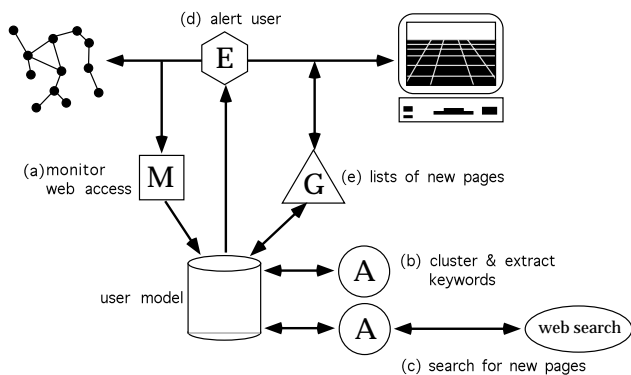


Figure 1: A confederation of agents that (a) monitors web access, (b) clusters retrieved pages and extracts keywords to identify clusters, (c) searches for new pages containing the keywords, (d) edits retrieved pages that are part of known clusters to point to alert user of related pages, and (e) generates pages containing lists of related URLs.

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 (Brooks *et al.* 1995). 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 the 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.

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, adding overhead. Using WBI, a single proxy which contains a series of separate transformations can be used instead.

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 (Pitkow & Kehoe 1996). For instance, web users have trouble re-finding pages that they have found before, and they become frustrated when servers are either down or subject to network delays. The basic set of functions we implemented include a personal history, shortcuts, and web traffic lights. We describe each of these in turn.

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 structure to allow rapid retrieval based on keywords (Salton & McGill 1983). 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.

The shortcut editor agent also relies on the personal history. Users often follow the same sub-sequence 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. 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.

One of our most popular editor/generator combinations is the *web traffic light*. This function annotates each HTML page with colored dots around each link 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 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.

Complex Agents Build User Models

Beyond the basic functions, we have implemented agents that build more complex user models to facilitate web browsing. One agent analyzes recently viewed pages to determine consistent trends in word usage. It then edits web pages to highlight links which seem to follow the pattern among pages the user is currently browsing (similar to Lieberman 1995). 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 1). We have also created a set of agents that allow users to annotate and group web browsing activity into useful books with minimal extra effort (cf. Card, Robertson, & York 1996). These books capture some of the thought and understanding which went into a particular searching or browsing session. These books can then be referred to later so that the abstraction underlying *web history* can be raised from single pages to reasoned sessions. We have also implemented a scheme for sharing books among users so that expertise in a subject area can be transferred.

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. This architecture enables rapid construction of agents that add arbitrarily complex functions to a user's web experience. We have implemented a number of agents that personalize the web for a user and that solve common web problems.

References

- Berners-Lee, T.; Fielding, R.; and Frystyk, H. 1996. Hypertext transfer protocol: HTTP/1.0. Technical Report RFC 1945, MIT.
- Brooks, C.; Mazer, M. S.; Meeks, S.; and Miller, J. 1995. Application-specific proxy servers as http stream transducers. In *Fourth International World Wide Web Conference*.
- Card, S. K.; Robertson, G. G.; and York, W. 1996. The webbook and the web forager: An information workspace for the World-Wide Web. In *CHI 96 Conference Proceedings*, 111-117. New York: Association for Computing Machinery.
- Lieberman, H. 1995. Letizia: An agent that assists web browsing. In *International Joint Conference on Artificial Intelligence*.
- Pitkow, J. E., and Kehoe, C. M. 1996. Emerging trends in the WWW user population. *Communications of the ACM* 39(6):106-108.
- Salton, G., and McGill, M. J. 1983. *Introduction to Modern Information Retrieval*. New York: McGraw Hill.