Supporting Cooperative and Personal Surfing with a Desktop Assistant

Hannes Marais and Krishna Bharat
Digital Equipment Corporation
Systems Research Center
130 Lytton Avenue, Palo Alto, CA 94301
Tel: 1-415-853{2157, 2137}
E-mail: {marais, bharat}@pa.dec.com

ABSTRACT
We motivate the use of desktop assistants in the context of web surfing and show how such a tool may be used to support activities in both cooperative and personal surfing. By cooperative surfing we mean surfing by a community of users who choose to cooperatively and asynchronously build up knowledge structures relevant to their group. Specifically, we describe the design of an assistant called *Vistabar*, which lives on the Windows desktop and operates on the currently active web browser. *Vistabar* instances working for individual users support the authoring of annotations and shared bookmark hierarchies, and work with profiles of community interests to make findings highly available. Thus, they support a form of community memory. *Vistabar* also serves as a form of personal memory by indexing pages the user sees to assist in recall. We present rationale for the assistant's design, describe roles it could play to support surfing (including those mentioned above), and suggest efficient implementation strategies where appropriate.

KEYWORDS: Desktop assistant, browserware, WWW, browser, annotation, asynchronous collaboration, community knowledge, bookmarks, indexing, barcodes.

1. INTRODUCTION

Surfing the web has become an all too common activity. While the web is rich in information, the pieces we seek are often sparsely distributed and hard to find. Hence, the substantial research effort in searching for resources on the web, visualizing search results, and homing in on the data that is actually relevant (e.g., [1, 2, 3]). The process of seeking out useful information on the web may be viewed from an ecological perspective. This metaphor has been used previously in the context of information foraging theory [3]. We invoke it to illustrate the need for asynchronous collaboration in search activities. Organisms (e.g., ants) tend to tackle challenging tasks, such as finding sustenance, by building for the future (amortization of work over time) and by dividing the labor (amortization over the group). The latter usually involves searching individually and leaving identification marks and trails to sources that others may follow. Similarly, in the context of information access, a user community with related interests and a willingness to cooperate (e.g., within a company or special interest group) could build community knowledge repositories, allowing the group to make efficient use of the information discovered. Besides providing exposure to individual findings, this process makes areas of interest within the group explicit, which is a form of information in itself. Of course, assimilation is only effective if the data that is found is any good. An integral part of surfing is finding useful documents and sensemaking.

In this paper we describe a desktop assistant called *Vistabar* that supports the kind of cooperative surfing described above. It is a continuously executing task on the desktop, and integrates smoothly with existing web browsers and browsing practices. It monitors user activity and provides services to find, understand and recall documents, and operations to assimilate them into a shared knowledge store. It is long lived and maintains state across browsing sessions. It is not tied to a specific browser and will latch onto whichever browser is currently active.

We prefer to call *Vistabar* an 'assistant' rather than an 'agent' to avoid raising expectations. While agent-like in many respects, the tool does not have an agenda of its own; nor does it model the user's beliefs. Hence we prefer the term the less proactive term 'assistant.'

Assistants such as *Vistabar* are strategically positioned between the user and the browser and are capable of interacting with both. This gives them a large amount of leverage, allowing them to be applied a variety of tasks, some of which may be application specific. One of the

goals of this paper is to motivate assistants that work with users within their surfing context. We term such "browser aware" applications *browserware* and expect to see an increasing number of them in the future.

To better understand the potential of *browserware*, let us look at what such assistants are capable of doing.

Firstly, they can control the browser independently, allowing them to replace the user as the driving force in the browsing process. This permits the use of the web as a presentation medium (as in [4]).

Also, as information is channeled into the browser the user assistant can help:

- Remember pages that were seen, to facilitate recall and build an interest profile.
- Automate common tasks such as logging onto a service.
- Transform the document (by augmenting or simplification).
- Share state with other users and services.
- Analyze and help understand the content.
- Relate to other entities within and outside the web.

If the document is active and contains an applet, the browserware component could be made application specific and can do much more. The component could serve the applet by maintaining persistent state, thus serving as the long-term memory of the applet, and by providing external control over the browser. A discussion of how such content might operate and the resulting security implications is a future research topic and beyond the scope of this paper.

We have two main contributions in this paper: (a) we motivate browserware and provide a general-purpose architecture for it, (b) we apply this technique to our specific problem, namely to make it easy to find, understand and assimilate information on the web in a cooperative way.

The rest of the paper is laid out as follows: Section 2 describes the rationale for the specific design we chose for *Vistabar*. Sections 3 and 4 describe how the assistant supports the two activities of locating information that is relevant and building a shared knowledge structure out of it. Section 5 presents some implementation details and Section 6 discusses related work, which is followed by a section on conclusions and future work.

2. DESIGN

We begin by considering some of the features that would be needed in a browserware tool. It should be persistent across sessions and be readily accessible on the desktop (should have a UI of its own). The tool should be in a position to monitor user activity (detect pages being viewed etc.) and control the browser when appropriate. It is desirable that the assistant be both logically and physically attached to the browser, even using it as a display modality (homogenous integration of UI).

Next, we look at some of the options available to build browserware and see how well they support the above features.

2.1 Design Options

The options, in decreasing order of involvement of the browser, are: custom browser, plug-in, applet, parasite and proxy. Their respective strengths and weaknesses are summarized in Table 1.

Table 1. Various Design Options

	Custom Browser	Plug-in	Applet	Parasite	Proxy
Control over Browser	Good	Fair	Poor	Fair	None
Monitoring	Good	Poor	Poor	Good	Limited
Persistent Presence	Within one Browser	Poor	None	Good	Good
Own UI	Not Applicable	Fair	Fair	Good	Good
UI integration	Not Applicable	Good	Good	Fair	Poor
Extensibility	Good	Not Applicable	Not Applic.	Good	Good

A custom browser would give the maximum possible access to control and monitoring. Persistence is restricted to the browser in question, since browsing with other browsers will go unnoticed. Extensibility is typically limited to the authors of the browser. Building a custom browser is a luxury that few software authors can afford. Browsers have become too complicated and too much of a moving target to justify re-implementation.

Plug-ins (or software components) are another option. We have seen this kind of extensibility other domains – e.g., graphic design tools and interactive development environments (IDEs). Both are extensible with third party components (e.g., Adobe Photoshop plug-ins [5] and Visual Basic controls [6]). Web browsers already support certain types of extensions (e.g., Netscape plug-ins [7]). Applets may be viewed as display oriented plug-ins with restricted access to the browser. Both of these tend to be launched over the network and are meant to operate within the context of a given page. Carrying state across pages is the responsibility of the web-server, which makes it site-specific at best. Due to their ephemeral nature plug-ins and applets are not in a position to support long-lived activities that may extend over a surfing session or multiple sessions. It is worth noting that due to the existence of multiple browsers, with various incompatibilities and tradeoffs, it is desirable that desktop assistants be associated with the user rather than a specific browser. Plug-ins and applets are tied to individual pages and do not present a clean way to maintain a permanent presence on the desktop.

Proxies and parasites are the remaining options.

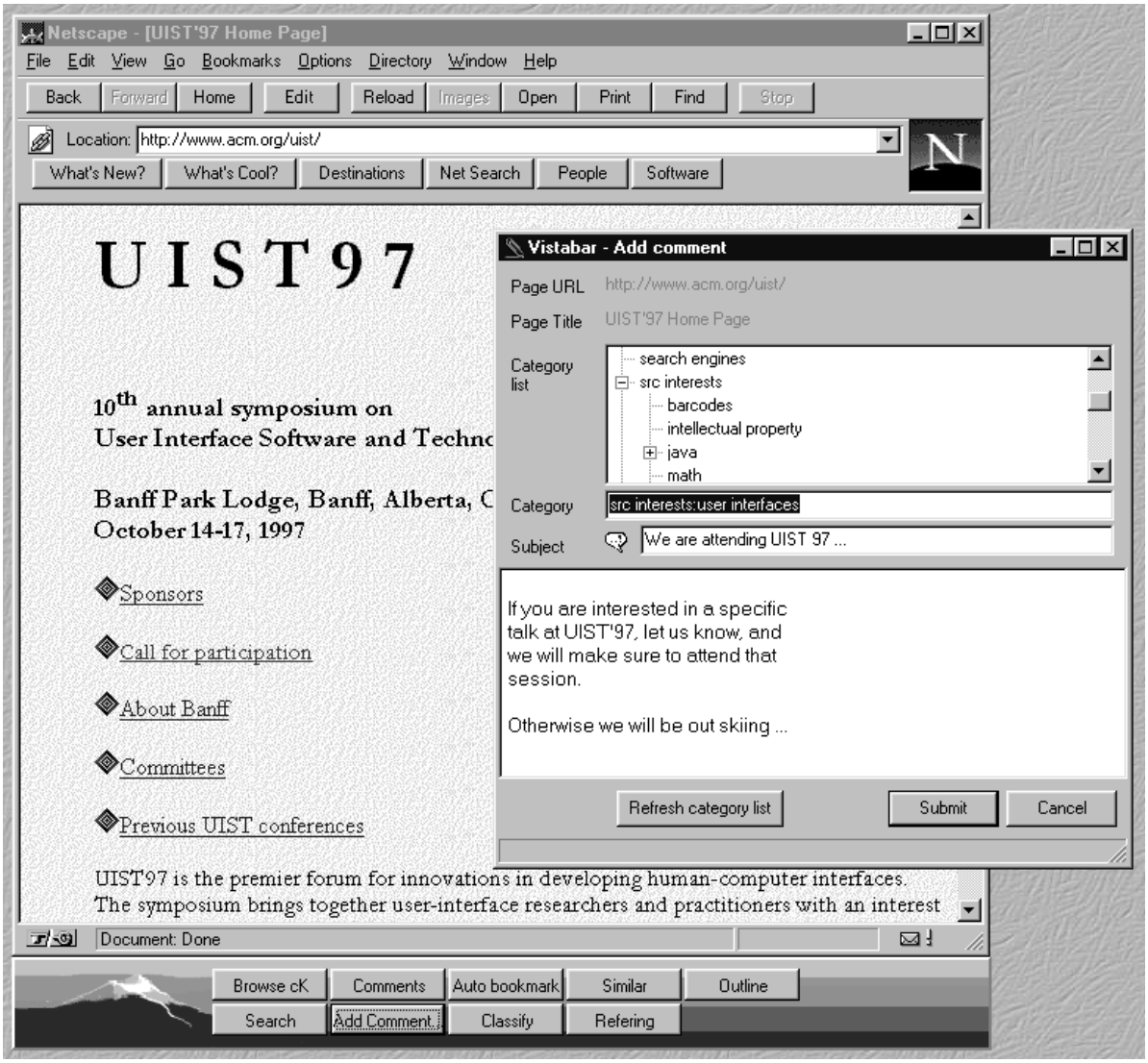


Figure 1. *Vistabar* attached to Netscape. The user is adding a comment to the UIST'97 home page.

A parasite is an application that attaches itself to another executing application and is able to monitor and control it through a published API. Familiar examples of parasites are debuggers. Knowing where the windows of the host are located, a parasite can wrap its own UI around it for better integration.

Proxies are more passive. They act by interposing in the communication between the browser and the web-server and are in a position to transform the data as it comes through or fetch it from alternate sources. Proxies provide more direct control over content and can be very effective for filtering and augmenting tasks. However, due to the presence of internal caches they cannot guarantee to successfully monitor what the user is looking at. Nor can they drive the browser's display. Also, we would like to physically situate part of the *browserware* UI within the browser by adding a control bar. A proxy can do this by augmenting the HTML that comes in. However, this presents some problems in practice. Adding content to a page that may in part be computed on the fly by an embedded script (in say, Javascript) is error-prone. Also, scrolling would cause the control bar to vanish. A workaround would be to place the control bar in a separate frame. This adds an artificial level of indirection that tends to interfere with navigation and bookmarking.

Although proxies seemed simpler and portable, they lacked some of the features we needed and imposed a constant overhead on browser operation which could be avoided. Hence, we chose to go with a parasitic architecture.

2.2 A Parasitic Model for Browserware

Here we describe the organization of a surfing assistant as a parasite. This design was used in the implementation of *Vistabar*, a browser parasite on Windows which works with both Netscape (versions 3.x, 4.x) and Internet Explorer (versions 3.x). Figure 1 shows *Vistabar* running as a browser parasite. The Windows specific details of the implementation are presented in Section 5.

In this model the assistant is constantly running and is accessible on the desktop. In our case this took the form of an icon on the Windows tray. It can be started and terminated without affecting browsers in any way. *Vistabar* tracks the currently active window on the desktop and wraps its user interface around it if it happens to be a browser it recognizes. This takes the form of a task-bar, which is either appended at the bottom (see Figure 1) or sits on the window's border (this is the only option when maximised). In addition to its own dialogs, the tool often creates task-specific HTML and directs the browser to display it.

Logically, the assistant consists of three parts: *Vistabar* (the front-end), *LocalKnowledge*, which is the user-specific memory of the assistant, and *CommonKnowledge*, which may be regarded as community memory. *Vistabar* makes use of the local disk to store its own persistent state. Shared persistent state resides on a server shared by members of the community. There are other servers associated with specific services which *Vistabar* needs to know about. The tool's configuration decides layout preferences and the servers to use.

3. FINDING INFORMATION ON THE WEB

In this section of the paper we are concerned with supporting the lone surfer's activities. We sketch five situations a web surfer seeking information might encounter, and the support a surfing assistant might render in each case. We do not claim the list of situations is exhaustive, and better solutions may exist. Our intention is to give the reader a feeling for the role a surfing assistant might play.

Figure 2. Searches on personal history can be highly effective

3.1 Supporting Recall

Web surfers often find themselves in situations where they recall having seen something on the web but cannot remember where the information was found. They might forget to bookmark an important page, or spend much time trying to retrace their steps to a page seen earlier in the session. Bookmarking everything that may potentially be useful leads to an unmanageable list of bookmarks. Also, it does not account for hindsight.

Our solution is to continuously index the full text of all viewed pages. Continuous indexing is feasible even on a modestly equipped PC. An avid surfer who encounters a 100 new documents per day (not counting duplicates), would encounter about 260 MB of raw text each year (figures based on AltaVista crawl statistics). After detagging and indexing this would reduce to a third of its size – less than 90MB. Given the steady growth in personal computer hard-disk size, this is fast becoming a reasonable investment for a year's 'total recall.' A full-text index is quite powerful because it can help recover a sketch of the document even if the actual page is missing.

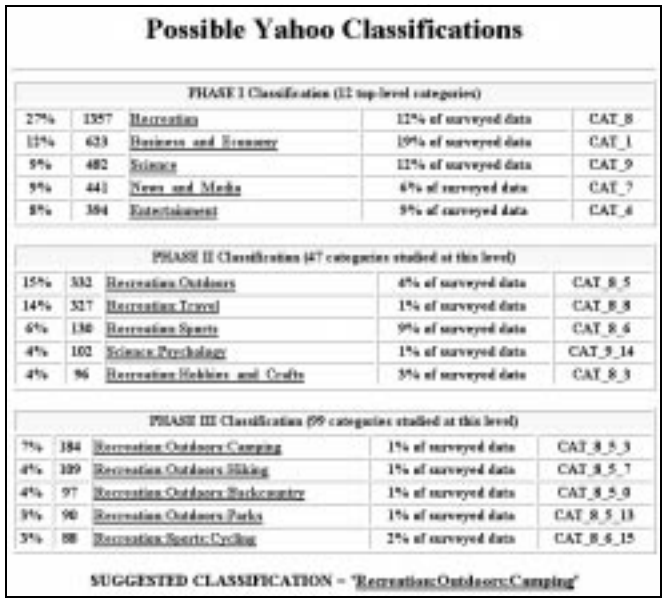
The *Vistabar* continuous indexer uses the NI2 library, the basis of the AltaVista search engine [8]. The pages the surfer reads are indexed transparently by a background process. At any time the user can perform a query on the index using the NI2 query language. This language resembles AltaVista's "Advanced Query" syntax, and supports operators like AND, OR, and NEAR, with keywords and phrases. Figure 2 shows the *Vistabar* query form, the results of a query, and a browser page launched from the query results.

Controlling the amount indexed. The default mode involves indexing all visited pages and tends to collect a large number of irrelevant pages in the index. Besides increasing the index's size, this tends to list pages that the user never actually read and does not recall.

We have a couple of modes to help cope with this:

- Explicit Mode: where the user needs to click on the 'Remember' button to index a page.
- Temporal Mode: where the page gets indexed if it was in view for a stipulated time (say 10 seconds).

In addition, there is an option to specify patterns that will prevent a URL from being indexing, e.g., result pages from search engines and pages from sites that change their content regularly.



Possible Yahoo Classifications

PHASE I Classification (13 top-level categories)

27%	1357	Recreation	12% of surveyed data	CAT_8
15%	633	Business and Economy	19% of surveyed data	CAT_1
9%	482	Science	12% of surveyed data	CAT_9
9%	441	News and Media	6% of surveyed data	CAT_7
8%	384	Entertainment	9% of surveyed data	CAT_6

PHASE II Classification (47 categories studied at this level)

15%	332	Recreation:Outdoors	4% of surveyed data	CAT_8_5
14%	317	Recreation:Travel	1% of surveyed data	CAT_8_8
6%	138	Recreation:Sports	9% of surveyed data	CAT_8_6
4%	102	Science:Psychology	1% of surveyed data	CAT_9_14
4%	96	Recreation:Hobbies_and_Crafts	3% of surveyed data	CAT_8_3

PHASE III Classification (99 categories studied at this level)

7%	184	Recreation:Outdoors:Camping	1% of surveyed data	CAT_8_5_3
4%	109	Recreation:Outdoors:Hiking	1% of surveyed data	CAT_8_5_7
4%	97	Recreation:Outdoors:Backcountry	1% of surveyed data	CAT_8_5_0
3%	90	Recreation:Outdoors:Parks	1% of surveyed data	CAT_8_5_13
3%	88	Recreation:Sports:Cycling	2% of surveyed data	CAT_8_6_15

SUGGESTED CLASSIFICATION = "Recreation:Outdoors:Camping"

Figure 3. Results from classifying the page in Figure 2

Building the index. Although NI2 is capable of continuously updating the index as new pages are seen, we prefer to batch updates over a set of documents. *Vistabar* maintains a pending cache of visited URLs. As soon as the number of cache entries exceed a user-defined threshold, or after a period of 10 minutes of inactivity, the pending pages are indexed and the cache is flushed. Indexing involves fetching the pages from the browser's internal cache, or directly from the Internet if not present. We delegate the responsibility of fetching to the browser, and hence passwords and cookies are handled automatically. The HTML is parsed and words are extracted. To save additional disk space, common words are removed.

For pages that warrant indexing, we check if the page has changed since it was last indexed, using a 64 bit digest or fingerprint calculated over the text. Fingerprints are associated with each indexed document. If a fingerprint repeats the document is not indexed. Otherwise, we remove the old document from the index, if present, before adding the newer version. Deletion is done in a mark and sweep fashion. The index is updated in batches. A background task periodically purges marked documents, adds new documents and does index reorganization. In the future, we would like to associate a time-to-live window with the index entries to avoid recalling very old documents. The downside of batched indexing is that very recent documents are not visible, but this is not a problem as long as the user is conscious of it.

3.2 Finding Related Information

Often, finding an interesting page is the starting point of a more extensive search process. The user might search further locally or look in places where related pages might occur. Usually the latter involves querying a search engine or looking within the appropriate category in a hierarchical classification service such as Yahoo or Excite [9,10].

Vistabar helps find related pages using both types of services.

Finding similar pages. The 'Similar' button causes *Vistabar* to invoke AltaVista with a query constructed from an analysis of the words in the document. Our simple scheme involves using the five most frequent words in the text of the document, excluding a list of stop words that occur with high frequency in English. This works remarkably at times and poorly just as often. A better scheme would take into account the expected frequency of words on the web. We intended this to be a lightweight operation with a quick response time.

Finding relevant categories. The 'Classify' button tries to lead the user to the most appropriate category for the document in question. We used Yahoo for this purpose. Our classification algorithm is general-purpose and works on any classification hierarchy including our own, as we see in Section 4. The classifier identifies and ranks the top five categories at each of the top three levels in the Yahoo hierarchy, selecting 15 out of roughly 1300 candidates (see Figure 3). This allows the user to start exploring at various levels in the tree. When a document has multiple topics the user is presented with alternate viewpoints, which can be quite beneficial.

The classifier uses a database, which was constructed in a preprocessing phase. We built profiles at each level in the Yahoo hierarchy going all the way down to the leaves. Category profiles were merged cumulatively from the leaves of the tree to the root. This allows matching to happen hierarchically. As is typical in information retrieval, we weight words by their frequency in the document relative to the whole corpus and do vector-space matching.

Instead of naively matching a web page against all categories in our database, we exploit the hierarchical nature of the classification tree to prune the number of comparisons. The profile matching process then involves following a path from the root of the tree to the required depth. When a specific category is identified as relevant, the search extends over its sub-categories. Since the discrimination ability of profiles diminishes upon grouping, a set of promising paths is followed rather than a single path. The database supports classification to any depth although we presently use 3 in the interests of response time.

Our implementation requires a substantial amount of data to keep track of all of the Yahoo categories and their associated word lists. Consequently we placed the classification service on a separate server which is shared by all *Vistabar* clients. *Vistabar* calculates a profile for the viewed document, which is then shipped to the classification server for analysis.

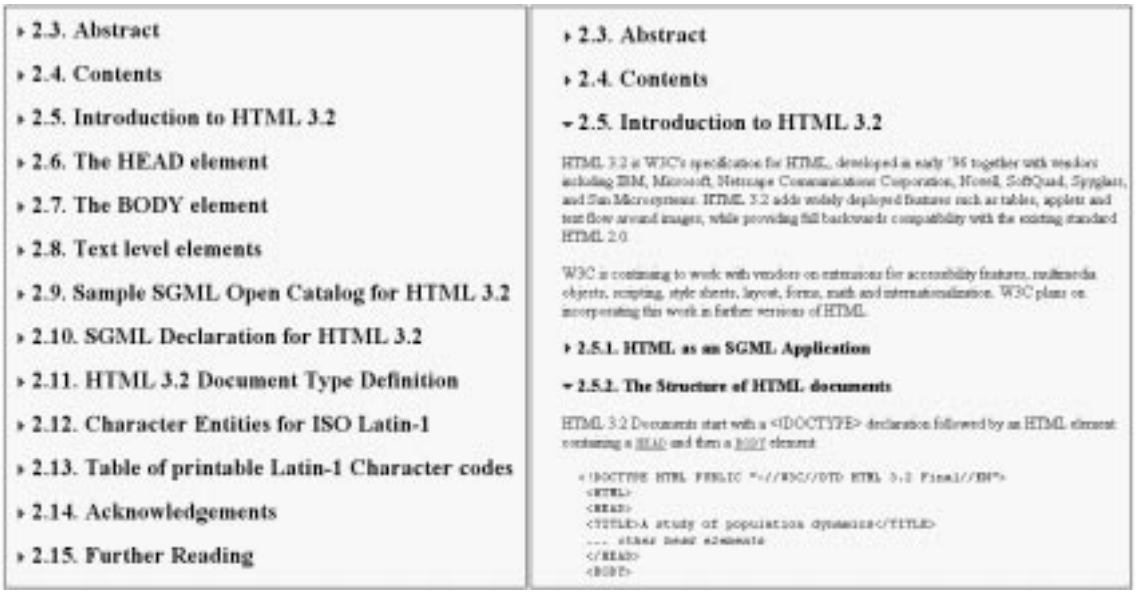


Figure 4. Zipping example - overview (left) and expanded view (right)

3.3 Supporting Focus in Context

It is not uncommon that a surfer finds a long web page that has not been formatted for hypertext display. These pages often come from legacy papers or reports converted to HTML without splitting the document into more manageable chunks connected with hyperlinks. *Vistabar* supports reading these pages by converting them into an outline format viewable in a standard web browser through a process we call "zipping."

A zipped web page contains the same information as the original web page except that the reader is provided with a way to expand and collapse nested sections of the document in a manner similar to an outline editor. The sections are delimited by headings identified by `H1, H2, …, H6` tags in the HTML. The depth of the section is determined by the start heading, which also decides its nesting. Zipping a page involves adding icons called zippers to each heading. A zipper indicates the state of a section (expanded or collapsed) and can be clicked to collapse or expand that section.

On finding a web page suitable for zipping, the surfer can use the 'Outline' button on the *Vistabar* to zip it up. The view presented by resulting page shows only the top headings in the page. At this stage the user can selectively open up interesting headings and watch the resulting modifications happen immediately on the browser's display. Uninteresting sections can be folded away in a similar way. Figure 4 shows a sequence of snapshots illustrating navigating inside a large document with zippers.

There are several ways to implement zipping; an overview of implementation alternatives and tradeoffs is found in [11]. *Vistabar* uses a novel approach that rewrites a web page as a script that executes inside the browser to render different views of the page. We use JavaScript [12], which is supported by the two most popular web browsers. The basic idea is as follows. As the JavaScript version of the page executes, it "prints" a particular view of the web page using a vector that maintains the state of each heading. A new view is generated by modifying the state vector and re-executing the script (achieved by reloading the page). We keep track of the state vector between successive views using a local cookie [13].

3.4 Finding Referring Pages

An interesting way to find out what people think about a web page or web site is to exploit the hyperlink structure of the web to find pages that refer to the page in question. In this way you often find either useful collections of links to similar topics or web page critiques. As the incoming links to a page are not explicitly known, we have to enlist the help of a search engine such as AltaVista that support *link* queries. The result of a link query lists all pages that refer to a specific page.

Vistabar provides a macro-like function that submits the currently viewed URL in a link query to the AltaVista search engine, allowing the surfer to surf backwards. We can imagine a useful set of macros to automate other tedious operations. Another useful function might be to fill out a form that asks for personal details.

3.5 Real World Associations

How does one make the connection between a real world object and corresponding content on the web? If a URL is the identification of a resource in cyberspace, then perhaps a barcode is the universal locator of an object in the real world. Many artifacts like books, CDs, groceries, and shipping bills now have barcodes. In addition there is an established industry associated with the allocation, printing,

Figure 5. Scanning barcodes with *WebMark*

and scanning of barcodes.

Our idea was to link objects in the real world with their counterparts in cyberspace by linking barcodes to URLs. This would allow us to look-up electronic "annotations" of a specific object in a web database, which would maintain pairs of barcode numbers and associated URLs. We call our realization of this idea, *WebMark*.

Our implementation consists of a public, web-based database of (barcode, URL) pairs, a low-cost hand-held barcode scanner, approximately the size of a credit card (Figure 5), and a downloading station interfaced to *Vistabar*. After collecting a number of barcodes, the surfer inserts the scanner into the download station, which then prompts the *Vistabar* to connect to the database to retrieve the annotations of the scanned objects.

We can imagine several uses for *WebMark*. Manufacturers could make product literature readily available using barcode associations. Anyone could extend the database with additional annotations – not just those who create the artifact. This would allow for user feedback to be tied to objects, allowing opinions and anecdotal data to be shared within a community.

Imagine finding on the web what other people think about a book you read, or finding an online copy of the VCR manual you threw away and now regret. While such requests could be answered by a text indexing service, the advantage of searching for a barcode rather than the name of the product is that barcodes tend to extremely specific and can be valuable when textual descriptions are difficult to produce. They usually refer to precise versions of the artifact, which is useful, say, when ordering replacement parts for a certain model. Also, they can be attached to objects that are difficult to describe or do not have any written markings on them (such as decorative pieces).

There have been other approaches to address this type of application, for example the real-world augmentation work done at Hitachi [14]. The strength of our approach is its easy implementation, based on off-the-shelf components, and the ability to leverage the large, global investment in barcodes. We imagine people collecting object references using either portable, key-chain sized scanning devices with memory for barcode storage, or small PDA-like devices with a scanner built in.

4. ORGANIZING INFORMATION IN A COMMUNITY

In this section we turn our attention to groups of surfers cooperating with each other to build a locally relevant shared database of information on the web. *Vistabar* helps users catalog articles and add comments to the knowledge store, and keeps users informed of others' opinions as they browse the web.

4.1 The need for "Common Knowledge"

It has been observed that one of the shortcomings of the many search services on the web is that they are too democratic [15]. Essentially everything is being indexed or collected, irrespective of value. Even when human selection is involved (as in Yahoo), the person in question may not be adequately knowledgeable in the area you are interested in, or there may be no category corresponding to your interests. This is the reason why scholarly publications are valuable: low quality information is filtered away by qualified reviewers and editors who are entrusted the responsibility of making decisions on our behalf. Unfortunately there are no practices to do the same thing on the web.

In research institutions, such as our own Systems Research Center (SRC), there are people with varied but often overlapping interests. These are people whose opinions and recommendations we attach a lot of importance to. E-mail and news postings are the traditional channels for such communication. We felt that an explicit mechanism to share findings on the web would be useful, and devised a

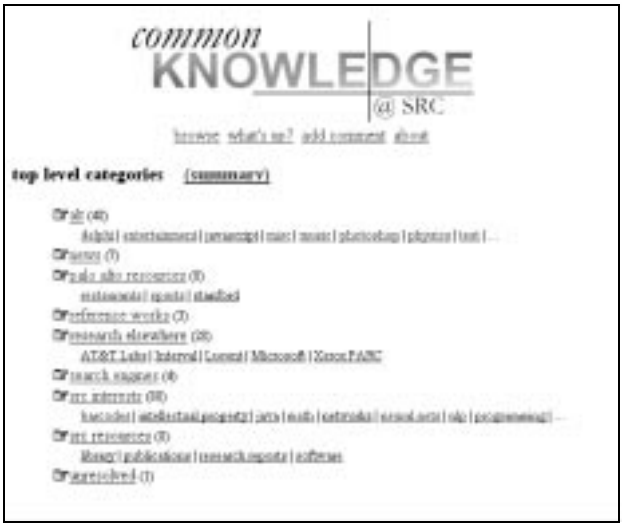


Figure 6. The SRC classification tree

system called *CommonKnowledge*, housed in *Vistabar*.

CommonKnowledge unifies the notions of bookmarking and annotations within the same framework. A bookmark assigns a category to a document; an annotation adds a comment. Both may be viewed as attributes of the document.

An annotation is usually tied to a specific portion on the document. We have noticed that people often do not use it in this manner, especially if the document was not locally authored. References to section numbers or headings often prove adequate. Hence, we chose a simpler approach: in *CommonKnowledge* comments may be associated with a document or a site but not with a part of a document. In the case of a site-annotation the comment is visible on all pages at the site. This informs surfers of "local landmarks."

The categories employed in *CommonKnowledge* are chosen by the users themselves. The category hierarchy forms a tree, although an item may be bookmarked under multiple categories. Figure 6 shows part of the current SRC hierarchy. A certain category tree was created to start the process, and got extended as needed.

When the user is unable to identify a category or is in a hurry, the 'Auto Bookmark' feature is invaluable. It matches the document against the profiles of the categories currently in the system and places it tentatively in one or more categories that might be appropriate. The tentatively bookmarked entries are confirmed later by other users. This creates an interesting division of labor: a user identifies the document as interesting, *Vistabar* chooses categories, and other users validate the categorization.

4.2 *CommonKnowledge* Functions.

CommonKnowledge supports several ways of accessing the shared bookmark structure. It consists of a central repository of "comments" that are associated with web pages and web sites. A comment might be as simple as an indication that a specific page is interesting, or a lengthy discussion about the contents of the page. A comment has many parts as is explained below. Note that comments differ from other services like e-mail and news in that they have a precise format that allows us to view and organize them in several different ways

The *Vistabar* and web browser taken together form the UI for the *CommonKnowledge* database. In combination, they implement (see Figure 1):

- An 'Add Comment' form that allows the user to quickly attach a comment to a page. The dialog allows them to select from a set of predefined categories or devise a new one. New categories are formed by textually defining a new path through the tree using ':' separators.
- A 'Comments' button to access the comments attached to a page and site. Comments cause the button to light up in red. We have an efficient scheme to detect comments, described in section 4.3.
- An 'Auto Bookmark' button to assign pages to categories on the fly, as described earlier.
- A *CommonKnowledge* browsing facility for exploring categories, generating summaries of categories, and accessing comments according to their time of insertion into the system. The browsing user interface is not a part of the *Vistabar* UI but uses the web browser instead (see Figure 6).

Comments. A comment consists of:

- The URL and title of the commented page.
- A short subject line related to the comment or the web page itself.
- An optional long discussion consisting of plain text.
- An identification of the category the page belongs to,
- An icon indicating a nature of the comment (e.g., humorous, informative, critical). Users cycle through a small sequence of icons and choose one. When a category assignment is considered tentative it is marked with a question mark.
- Miscellaneous fields such as author, date, etc.

After the surfer completes the details of a new annotation, all fields except the long comment are submitted over HTTP to the *CommonKnowledge* server. The long comment is uploaded with FTP to a web server that the user normally uses. The advantage of using local web servers for long comments is that the system scales well. It can accommodate a large organization where users tend to use different servers for their personal pages but wish to use a single *CommonKnowledge* server.

For automatic bookmarking we use the same scheme employed for classifying pages under Yahoo categories. The *CommonKnowledge* server maintains profiles for various categories. New bookmark entries are processed in batches and category profiles are incrementally updated. When a bookmark is being assigned and a good match is not available, the page is added to an 'Unresolved' category. This indicates that it will need to be explicitly bookmarked by someone in the community.

4.3 Detecting Comments Efficiently with Bloom Filters

A naive implementation of the comment indicator would repeatedly poll the *CommonKnowledge* server to check if the visited page has comments attached. This technique scales poorly and has the drawback that all browsing

activity is reported to a remote server, raising privacy concerns. We chose an approach that does not require a database lookup at all.

We use a fast set membership test based on a Bloom filter [16]. A Bloom filter is an array of bits used to represent a set of elements that do not admit a linear ordering (i.e., there is no way to compute a unique index value from a set element). In our case, the set of URLs with comments associated with them would form such a set.

Given a Bloom filter *m*, an element *e* is inserted into *m* by hashing *e* with *k* different hash functions H_i, *i=1.. k*, and setting the corresponding bits, $m[H_i]$. Checking if an element belongs to the set involves hashing the element again as described above, and checking if *all* the corresponding bits are set. This is a conservative test, and it is possible to obtain false positives. The accuracy depends on the size of *m* and *k*. In the case of a database with 20,000 commented pages and *k* set to 4, a 32 KB Bloom filter has a less than 1% probability of a false positive. This low error rate is acceptable for our application.

The Bloom filter is maintained by the *CommonKnowledge* server and periodically downloaded by *Vistabar*. *Vistabar* consults the local copy of the Bloom filter to determine if a page has a comment or not. This scheme scales well, as it does not require network access or continuous database lookup.

5. IMPLEMENTATION

Vistabar runs on Windows 95/NT with recent versions of Netscape and Internet Explorer. Tight integration of *Vistabar* with the web browser is possible by the browsing APIs (using Dynamic Data Exchange) supported by Netscape and Microsoft [17,18]. Unfortunately this API is limited to communication with short string-based messages and is thus quite limited. The bulk of *Vistabar* and *CommonKnowledge* is built with Delphi 3.0, Perl, commercial web servers and SQL databases.

6. RELATED WORK

Intel's Internet Telephone and DocuMagix's HotPage [23] are examples of browserware applications, that can attach themselves to the active web browser.

Lieberman's Letizia [25] is a user interface agent that tracks user browsing activity and anticipates future items of interest by doing concurrent, autonomous exploration of links from the user's current position. Letizia's level of control over the browser is similar to Vistabar. It uses AppleEvents and AppleScript to communicate with Netscape running on the Macintosh OS. Similarly, WebWatcher [26] is a "tour guide" agent for the World Wide Web. Once you tell it what kind of information you seek, it accompanies you from page to page as you browse the web, highlighting hyperlinks that it believes will be of interest. Its strategy for giving advice is learned from feedback from earlier tours.

A popular approach to extend a web browser with application specific functions is with proxies [27]. The W3 Surf Navigator [19] is a proposal for a surfing assistant that gathers, indexes, and visualizes viewed documents. WBI [24] is a proxy-based system for performing personalized transformations to web pages based on past history. It performs useful tasks such as adding shortcuts to common paths and annotating links with network speed information.

There are several systems for annotating web pages. The ComMentor system [20] is an inlining annotation system based on a modified web browser that can also be employed for shared bookmarks. It organizes annotations as sets that the user can subscribe to. There is no hierarchical organization for annotation sets. In a similar system [21] the custom browser is replaced with a proxy. Both systems poll annotation servers for each page visited.

Clustering approaches have been applied for organizing bookmarks into categories [22]. Rather than building category profiles they make use of inter-document similarity for computing new categories.

InteractiveDESK [14] is a system for linking real-world objects to electronic documents in a database. They use video analysis for object recognition. This is resource intensive, potentially error-prone and cannot make fine distinction between versions, as one can with barcodes.

7. CONCLUSIONS

We use the term *browserware* to denote a class of applications that are situated between the user and web-browsers and interact with both. They work within the user's browsing context to enhance the surfing experience in various ways. We considered various implementation schemes and chose a parasitic organization because it integrates tightly with the browser without requiring implementation of a custom browser.

Vistabar is a realization of a surfing assistant using our parasitic model. It supports information gathering and organization with a suite of useful tools, both in the context of the individual surfer and within a community that wishes to leverage the findings of individuals. Novel aspects of *Vistabar* include continuous indexing of personal surfing history, efficient detection of annotations, automatic classification of web pages, and annotations to real-world objects.

Our experience with *Vistabar* has shown us that this design is rich enough to support a variety of services. In particular we are excited by the prospect of supporting dynamic, site-specific behavior in this framework. We are currently looking at a dynamically extensible architecture for browserware based on downloadable agents. This would allow content-providers to associate application-specific assistants with their web pages and add a new dimension to publishing on the web.

8. REFERENCES

1. Mukherjea, S., Foley, J.D., and Hudson, S. Visualizing Complex Hypermedia Networks through Multiple Hiearchical Views, In proceedings of *CHI '95*, pp. 331-337.

2. Baldonado, M.Q.W., and Winograd, T. SenseMaker: An Information-Exploration Interface Supporting the Contextual Evaluation of a User's Interest, In proceedings of *CHI '97*, Atlanta, GA.

3. Pirolli, P., and Card, C. Information Foraging in Information Access Environments, In proceedings of *CHI '95*, pp. 51-58.

4. Yamaguchi, T., Hosomi,I., and Miyashita, T. WebStage: An Active Media Enhanced World Wide Web Browser. In proceedings of *CHI '97*, Atlanta, GA.

5. *Adobe*. Software Development Kit (SDK) for Photoshop, http://www.adobe.com/supportservice/devrelations/sdks.html

6. *Microsoft*. Visual Basic Control Creation Edition. http://www.microsoft.com/vbasic/controls/

7. *Netscape*. The LiveConnect/Plug-in Developer's Guide, http://home.netscape.com/eng/mozilla/3.0/handbook/plugins/index.html

8. *AltaVista Inc*. http://www.altavista.digital.com/

9. *Yahoo!* http://www.yahoo.com/

10. *Excite*. http://www.excite.com/

11. Marc H. Brown, Hannes Marais, Marc A. Najork, William E. Weihl. Focus+Context Displays of Web Pages: Implementation Alternatives. *WWW6 conference poster*, http://poster.www6conf.org/poster/745/poster745.html

12. *Netscape*. JavaScript 1.1 Language Specification. http://home.netscape.com/eng/javascript/index.html

13. *Netscape*. Persistent Client State: HTTP Cookies. http://home.netscape.com/newsref/std/cookie_spec.html

14. Arai, T., Machii, K., and Kuzunuki, S. Retrieving Electronic Documents with Real-World Objects on InteractiveDESK. Proceedings of *ACM UIST '95*, pp. 37-38.

15. Berghel, Hal. Cyberspace 2000: Dealing with Information Overload. In *Communications of the ACM*, 40, 2 (Feb. 1997), 19-24

16. B. H. Bloom. Space/time Tradeoffs in Hash Coding with Allowable Errors. In *Communications of the ACM*, 13, 7 (1970), 422-426.

17. *Netscape*. Netscape's DDE Implementation. http://home.netscape.com/newsref/std/ddeapi.html

18. *Netscape*. OLE Automation in Navigator. http://home.netscape.com/newsref/std/oleapi.html

19. Gerald F. Ehmayer. W3 Surf Navigator - an Intelligent Listener Agent. *WWW5 Workshop on Artificial Intelligence-based tools to help W3 users*, 1996, Paris, France.

20. Martin Röscheisen, Christian Mogensen, and Terry Winograd. Beyond Browsing: Shared Comments, SOAPs, Trails, and On-line Communities. *Third International World-Wide Web Conference* in Darmstadt, Germany, April 1995.

21. Matthew A. Schickler, Murray S. Mazer, and Charles Brooks. Pan-Browser Support for Annotations and Other Meta-Information on the World Wide Web. *Computer Networks and ISDN Systems*, 28, 7–11, 1996

22. Yoelle S. Maarek, Israel Z. Ben Shaul. Automatically Organizing Bookmarks per Contents. *Fifth International World Wide Web Conference*, May 6-10, 1996, Paris, France.

23. DocuMagix Inc. http://www.documagix.com/products/dhotpage.htm

24. Barrett, R., Maglio, P.P., and Kellem, D.C., How to Personalize the Web. In proceedings of *CHI '97*, Atlanta, GA.

25. Lieberman, H. Letizia: An Agent That Assists Web Browsing. Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI'95), Montreal, August 1995.

26. Robert Armstrong, Dayne Freitag, Thorsten Joachims, and Tom Mitchell. WebWatcher: A Learning Apprentice for the World Wide Web. AAAI Spring Symposium on Information Gathering from Heterogeneous, Distributed Environments, Stanford, March 1995.

27. Charles Brooks, Murray S. Mazer, Scott Meeks, and Jim Miller. Application-Specific Proxy Servers as HTTP Stream Transducers. Proceedings of WWW4, Boston MA USA, December 1995.