Web Navigation Architectures
Browser, **A**pplication, **S**erver or **E**mbedded?

Carsten Sørensen
c.sorensen@lse.ac.uk
London School of Economics and Political Science. United Kingdom &
Laboratorium. Trollhättan/Uddevalla University. Sweden

Daniel Macklin
Tony Beaumont
a.j.beaumont@aston.ac.uc
Computer Science, Aston University. United Kingdom

Abstract

The World Wide Web is increasingly becoming the preferred repository of information. The strength of this information infrastructure is also its weakness. Faced with the chaos of millions of places to go and thousands of places to remember having bee, the thousands of new Web users who join every day, need a helping hand. The aim of this paper is, by way of an experiment designing a prototype system, to conceptualise the architecture components of Web navigation support. The prototype supports the ranking of bookmarks based on monitoring user behaviour and recording user ranking. The BASE framework is, based on a survey of existing systems, suggested as a means of understanding the pragmatic technological choices. The framework is applied in characterising a number of current Web navigation technologies.

Keywords: Web navigation support, system architecture

1. Introduction

The World Wide Web (the Web) is exploding. The immense success of the Web as a repository of information on literally everything can not be disputed. The adoption rate of the underlying technology, the HTTP servers, increased from 200 in October 1993, over 1500 in June 1994 to an estimated more than two million in 1997 (Darken, 1998). The comparison with the introduction of Gutenberg's printing press, wireless radio and even the mighty television seems infinitely slow in comparison. Every day thousands of people connect to the Internet (the Net). In the United Kingdom it is estimated a growth of 10.000 new Internet users every day (The Observer, April 4, 1999). According to Wired (April 1999, p.70), 40% of new adult Net users in the USA have never attended college and 23% have incomes below $30.000 pr. year. Newcomers to the Net are in other words ordinary consumers, and not any longer only computer professionals and enthusiasts. Their primary windows to this information infrastructure are Email and the Web. Increasingly the infrastructure will be applied in a broad range of areas. Consumers will increasingly turn to the Net when they seek information, when they buy goods and services, and when they wish to connect with others. We can look at Finland to get an idea about where it is all going, where 10% the population use the Internet weekly to pay bills or buy services. Only a handful of other countries has 10% of their population connecting to the Net on a weekly basis at all (Lyytinen and Goodman, 1999). The Web has been the main driving factors in the diffusion of Internet technology to businesses and consumers. It provides the advantages of hyper-linked documents easily accessible through Web browsers. These Web browsers, which in the case of the two market leaders Netscape Navigator and Microsoft Internet Explorer, also provide access to electronic mail and to electronic bulletin boards. The Web browsers and a handful of Web directories and search engines, increasingly coined "Web Portals", thus presents the user with a "one stop" solution for accessing the infrastructure.

As argued by several, one of the main strengths of the Web, its lack of overall structure of the interconnected hypertexts, is also one of the main problems which must be addressed (Lieberman, 1995; Tauscher and Greenberg, 1997a; Tauscher and Greenberg, 1997b; Darken, 1998; Sørensen, 1998). Nielsen (1999) argues that based on the estimated growth in number of Web sites from 4 million early 1999 to around 200 million sites during 2003, we are facing problems in several areas. Nielsen, primarily concerned with the usability of the Web, argues that the explosive growth of Web sites will impose a huge challenge for the design of sites that can easily be accessed by everyone from everywhere. This involves adhering at least to basic principles for how to design Web sites that are easy to navigate. It also involves designing client software supporting navigation and visualisation of Web sites, despite their usability condition (Nielsen, 1999).

A majority of the research efforts in the area of Web navigation reflects issues related to searching for new Web resources. A just as important and related issue concerns the problem of maintaining repositories of up-to-date relevant indexes to the Web, once they have been located. For the casual and curious Web navigator, this will perhaps not be a main problem in the short term. The situation is, however, different for people who in their daily work use the Web intensively as a source for information and news. Once they over a long period have found out where they want to go, they will increasingly experience problems determining where they have been, and which of the places they have been are worth visiting again. Most users will only use the support provided by the Web browser they are using, which in reality only is the very expensive but structured bookmark facility and the unstructured but cheap history log (Sørensen, 1998). As argued by Nielsen (1999), the Web browser, although suited for navigation of a Web containing a few hundred million pages or less. With the present explosive growth, the current Web navigation technology is not adequate. The pull-down menu is, for example, an extremely inadequate means for managing bookmarks, and there must be better support for pruning bookmarks based on user behaviour (Nielsen, 1999).

The aim of this paper is to explore the technological choices involved in providing support for Web navigation. More specifically, the architecture of navigation support is investigated with a view not only to the use of the technology but also the overhead of installing it. Even though the majority of Web navigators primarily use a combination of web browser search engine, and bookmark file, a more sophisticated need will emerge. This need will be driven by the explosive growth of the world of links navigated.

Navigation support functionality can be implemented as one of a number of technical solutions, for example as a set of Perl scripts, a Web browser plug-in, a stand-alone application, a Java client-server solution, a Web-site server service. Each of these solutions will have distinct consequences for the end-user in terms of installation and use. Given that the technology increasingly will be in the hands of consumers and not professionals in organisations with available support staff, we must strive to understand the consequences the choice of architecture has on the use and maintenance of the technology. No research has so far looked at the relationships between the choice of navigation support architecture and navigation activities. In order to provide rich data on the systems architecture issues involved in Web navigation support, we developed a prototype system. The Java client-server system supports semi-automatic ranking of bookmarks based on recorded user behaviour and on explicit user-ranking of sites. The experiment resulted in the BASE framework characterising the basic architecture components of Web navigation support as a combination of the four main elements:

(1) Browser: Part of or the entire Web navigation support functionality is provided as an integral part of the Web browser
(2) Application: Part of or the entire functionality is provided in a separate application which is installed on the client computer
(3) Server: Part of or the entire functionality resides in a server. Access to this functionality can either be through a Web page or by a client application
(4) Embedded. Part of or the entire functionality is embedded in or part of a Web page downloaded, such as a Web browser plug-in, a JavaScript panel, or a Java applet

The following section presents a conceptual framework characterising the functional elements of Web navigation and discusses related disciplines. Section 3 presents and discusses the main issues encountered in an experiment designing a semi-automatic bookmark management system. Section 4 synthesises the findings from the experiment in a framework characterising the architectural elements of Web navigation support. Section 5 discusses different types of Web navigation technologies using the BASE framework and the Web navigation model. Section 5 concludes the paper.

2. Web Navigation

Supporting the navigation of ill-structured hypertext documents is a very complex task, especially if the ambition is to obtain a high degree of automation. This section characterises Web navigation in terms of the functional components and the challenges for providing computer support. This will provide an overview of the possible elements contributing to the complexity of the task.

Conceptualising Web navigation

In order to analyse how current and future Web navigation support match the activities of an individual navigating the Web, we need a set of basic concepts. The Web Navigation Model presented below provides an analytical model characterising the functional properties of Web navigation, i.e., characterising what actually goes on when a person navigates the Web (Sørensen, 1998). The aim of the framework presented below is to pragmatically characterise the functional elements. It is not the purpose, from a cognitive perspective, to precisely describe what goes on inside the head of the person navigating. Nor is it the purpose to focus on one particular allocation of functionality between human and computer in supporting navigation. Hence, the simple, yet powerful analytical model characterising the activities involved in navigating the Web as the following four functional components (Figure 1). Each function is exemplified by the use of an ordinary Web browser in conjunction with a search engine.

***Declaring**:* Defining or refining the declarative parts of the search and navigation profile. The user declares the search profile simply by entering a specific URL in the Web browser. The URL might be retrieved from the bookmark file or from the history log of recent Web resources visited. In both cases the management of the declaration history and context is left to the user.

Searching: Exploratory investigation of possible Web pages to download based on the declared profile. Once the user has used a search engine to locate a (large) number of URLs, he or she can choose a number of URLs to visit. Once a particular URL is visited, it might, in turn, lead to new lines of inquiry etc. etc. It will be entirely up to the user to perform the navigation. The browser will only download the hypertext pages.

Exploring: Actively choosing a particular path to investigate and analyse. Once the user have used a search engine to locate a (large) number of URLs, he or she can choose a number of URLs to visit. Once a particular URL is visited, it might, in turn, lead to new lines of inquiry etc. etc. It will be entirely up to the user to perform the navigation. The browser will only download the hypertext pages.

Evaluating: Assessing the quality of the information retrieved in relation to the existing body of knowledge represented by discarding undesirable, and systematising desirable information. Presented with the contents of a given page, the user then decides whether the information on the page is of interest and if any of the links ought to be pursued.

The outer arrows in the model (Figure 1) denote the main cycle, and the inner arrows represent the possibility to cut out one or more activities in a particular cycle. The allocation of functionality among human and computer is at this stage not determined for the activities. Hence, the model equally describes the manual use of a conventional Web browser and a more automated navigation scenario where advanced functionality replaces some manual activities. It is important to note that at any time in the navigation process the user may leave behind traces for subsequent inspection. These can for example be a profile declaring the user's interest, the recording of the navigation history, or the few selected bookmarks saved for later reference.

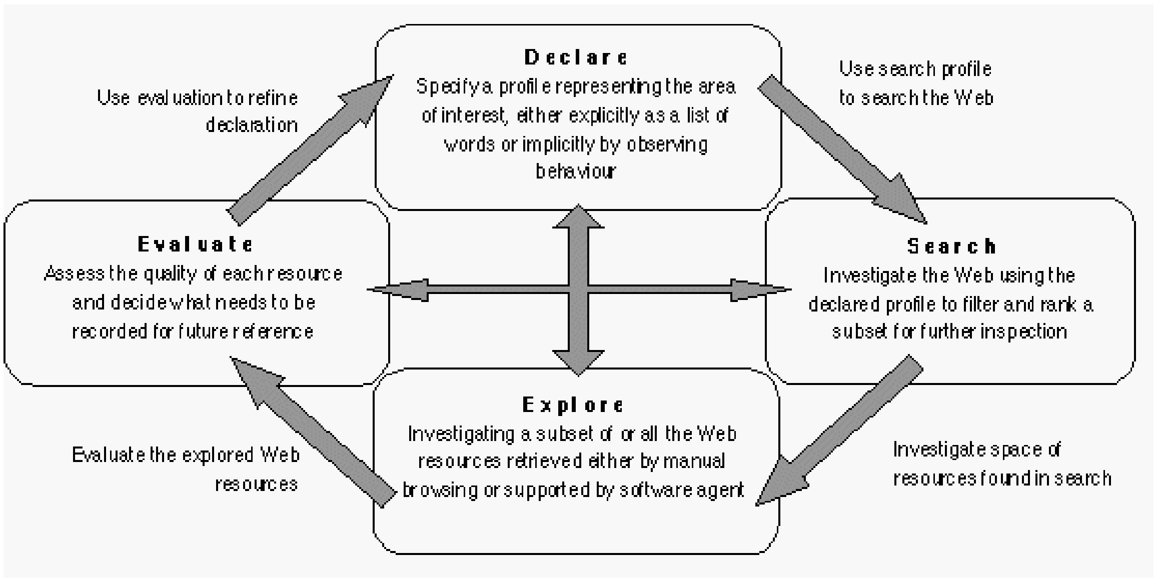

Declare
Specify a profile representing the area of interest, either explicitly as a list of words or implicitly by observing behaviour
Use search profile to search the Web
Search
Investigate the Web using the declared profile to filter and rank a subset for further inspection
Investigate space of resources found in search
Explore
Investigating a subset of or all the Web resources retrieved either by manual browsing or supported by software agent
Evaluate the explored Web resources
Evaluate
Assess the quality of each resource and decide what needs to be recorded for future reference
Use evaluation to refine declaration

Figure 1: Web Navigation Model characterising the activities involved in Web navigation (Sørensen, 1998).

Understanding Navigation

A number of basic disciplines and techniques provide input for designing computer support for Web navigation. The relative lack of structure of the Web implies that part of the problem relates to information retrieval and filtering. In cases of more structured data, it may be feasible to apply linear statistical techniques for data-mining, which is quite difficult on the Web (Etzioni, 1996). Oard (1997) characterises the issues in information retrieval and filtering as a combination of the change rates of the information need and the information source. In general, information retrieval and information filtering can be viewed as two sides of the same problem (Belkin and Croft, 1992). It is generally considered impossible to tackle situations where there both are a high information need and information source change rate. Since the Web is characterised by a high source change rate, support for information filtering must assume relative low information need change rate. This has also been termed "persistence of interest" (Lieberman, 1995), and relates to the assumption that the user has a consistent interest over a period of time. In order to represent this interest, the system may employ techniques from the area of user modelling (Allen, 1990).

As argued by Oard (1997); "Information filtering can be viewed as an application of user modelling techniques to facilitate information detection in dynamic environments." In order to filter and rank the resources found in a dynamic text corpus, the interest of the user can be applied as a filter. This relates to the field of user modelling where the challenge is to represent user preferences and record user behaviour. This information is subsequently used to support the navigation process. Precision and recall, the two traditional concepts from information retrieval, are not suited to describe information retrieval on the Web, since the concepts are based on the notion of a complete set of documents (Nielsen, 1999). The Web will in most cases contain far more relevant documents than anyone would have the time to inspect.

Oard (1997) characterises three fundamental information-seeking tasks: Collection, Detection and Display. Comparing these tasks to the functional model for Web navigation above, the tasks are orthogonal to the functions. The typical search engine will, based on a search profile, collect, detect and display Web resources as a ranked and filtered list collected from the search engine's index of a large proportion of the Web. Similarly, from the point of view of a user obtaining a list of Web resources from the search engine, the list represent resources collected. The exploration and evaluation of these represents the detection task. Saving the desired ones as bookmarks denotes the display task. Nielsen (1999) argues for the need for functionality supporting:

(1) Aggregation -showing a single unit that represents a collection of smaller ones
(2) Summarisation - representing a large amount of information with a smaller one
(3) Filtering by eliminating unwanted information; and
(4) Elision which is example-based representation.

Since the Web is a distributed and inter-linked global networks, it is cumbersome if not impossible to inspect and explore a substantial amount of it without the use of semi-automatic means. Software agents have become increasingly important as a technique providing semi-automatic access to the Web (Maes, 1994; Lieberman, 1995; Krulwich, 1997; Maglio and Barrett, 1997; Fagrell and Ljungstrand, 1998; Joshi and Singh, 1999). Web agents, which also are referred to as 'robots', 'wanderers', 'crawlers', or 'spiders', are programs that automatically traverse the Web's hypertext structure by retrieving a document, and recursively retrieving all documents referenced. The search engines to collect data for indexing also use web robots. A Web browser is not in itself a robot since it is operated by a human user and does not automatically retrieve referenced documents. The agents support some of the navigation tasks to be conducted semi-automatically, and a number of Web agent systems are available (see for example www.botspot.com). Schubert, Zarnekow and Brenner (1998) suggest a three-dimensional framework for classifying software agents in general. They suggest the dimensions:

number of agents (single agent or multi-agent systems); intelligence (simple or complex); and mobility (static or mobile).

Often people will not only seek resources on the Web, but also ask others for help. This has been formalised and supported in a number of systems and prototypes. Mutual recommender systems supports users in obtaining suggestions based on either the explicit preferences of other users or from the analysis of recorded behaviour of other users (Balabanovic and Shoham, 1997; Oard, 1997; Terven et al., 1997; Fagrell and Ljungstrand, 1998).

When looking at this complex picture of research fields and computer technology, it is clear that they offer a potential when addressing the complex issues of supporting the process of making sense of and navigating a chaotic and unstructured world. However, it is also clear that none of the individual research fields or areas of technology single-handedly can address the complex issues we are facing. It seems that people within the area of autonomous software agents tend to overemphasise the potential of that particular technology without addressing some of the potential drawbacks (Schneiderman and Maes, 1997; Joshi and Singh, 1999). Similarly, areas such as information retrieval, information filtering, user modelling, artificial intelligence, and distributed networks can all provide important lessons for the study of Web navigation support. A focus on the each of the fields as such instead of on the problem at hand, i.e., investigating the support for people who experience problems of managing their interaction with the Web, carries with it the danger of digress.

Although relatively simple, the Web navigation model presented above, marks a pragmatic starting point acknowledging the most important issues in the near future. It also provide a simple framework from which to understand the need for input from various disciplines by in each of the four functional elements discussing the allocation of functionality between the human and the computer. Declaring a search can, for example be supported by providing a profile of the user's interest, i.e., user modelling. This profile can either be derived from observing user behaviour or by explicit stipulation. Software agent technology can in conjunction with a user model be applied to semi-automatically traverse sets of Web resources and provide filtering of the outcome in order to support both the exploration and evaluation of Web resources. Supporting Web navigation is, in other words, not about sticking to a well defined discipline but about providing the right mix which both address the need for support without requiring extensive overhead for installation and training.

3. The Java Bookmark System

This section documents an experiment with the purpose of designing semi-automatic support for the exploration and evaluation of Web resources. The purpose of the experiment is to provide rich data on the architectural issues involved in designing Web navigation support. In the development of a prototype we have taken up one of the challenges promoted by Nielsen (1999), to let bookmarks reflect the history of the users navigation behaviour and the users explicit preferences. The prototype presented in this section supports the exploration and evaluation of Web resources by semi-automatic re-ordering of bookmarks according to the intensity of use and the user's indication of importance. One of the simplest, yet most widely used methods of recording interest from the Web is the humble bookmark file. The main reasons for its popularity are threefold. Firstly, it is a built in feature of almost all Web browsers, and hence readily available to anyone connected to the Web. Secondly it is incredibly easy to use. Finally the concept of book marking is intuitive to everyone. However despite these advantages, various research studies have pointed out that the standard bookmark file is inadequate at recording Web interest, and hence could be improved to help aid re-visitation (Tauscher and Greenberg, 1997b; Nielsen, 1999).

One of the main reasons for this inadequacy is the book mark file cannot be easily changed to keep up with the dynamic, constantly changing nature of the Web, which means that resources come into existence, are modified, and eventually may disappear. Another important reason is that the bookmark file cannot react to subtle changes in users' persistence of interest. These problems lead to users having to spend inordinate amounts of time amending and pruning their bookmarks. Unfortunately for most Web users the time involved is too great, and the bookmark file descends into a jumbled mess of URLs, which according to Taucher and Greenberg (1997b; 1997a) can be seen to be almost as inefficient as not having bookmarks at all. The aim was, therefore, to develop a system improving the basic bookmark file without burdening the user with greater complexity, functionality or computational effort.

The initial idea

The first stage in the project was to identify how to increase the flexibility of the bookmark file without compromising the factors that lead to the popularity of bookmarks. After initial investigations, the most promising solution was to develop functionality to re-order the users bookmarks in such a way that the URLs the user felt to be most Important and used the most could be found towards the top of the bookmark file. The resulting effect could be seen as analogous to conducting a bubble sort of the URLs in the users' bookmark file according to the users' persistence of interest. It was believed that by following this strategy numerous advantages could be gained. Firstly the basic model behind the book marking system could be left unchanged, in that when the user found an interesting resource they would add it to the bookmark file. Secondly the idea of finding the most important information at the top of a list, and the least at the bottom is as intuitive as the idea of book marking itself. Thirdly a system such as this would build in some flexibility into the bookmark file, to help it more accurately model a users' persistence of interest. Fourthly an re-ordering algorithm could be constructed that lead to a gradual, rather than sudden change in the order of the bookmark file, which could prevent the user becoming confused as to the location of their bookmarks.

Re-ordering bookmarks

With the model of operation set, a suitable re-ordering algorithm that took into account both persistence of interest and the dynamic nature of the Web, while not intruding too much into the basic operation of the bookmark file had to be designed. The obvious solution to this problem was to use a two-stage approach. Initially the bookmark would be arranged solely by how important a user felt a resource to be. This functionality was incorporated by allowing the user to give each bookmark an information importance score ranging from 0 – 100. The second stage would be to monitor how the user used the bookmark resources by collecting and storing various statistics. This information would be used to amend the initial information rating score, and the bookmark would be re-ordered. The best way to describe why this decision was made is to walk through a possible scenario of bookmark usage that we came up with. The scenario shows how the automatic re-ordering bookmark can save upon the time that a user would need to properly re-arrange their bookmark file. Initially a user finds two Web pages of different content, but similar topic. At the time, the user rates the first page very highly, giving it a score of 100, and the second one less highly giving it a score of 50. After the information rating, the first page is found at the top of the bookmark, while the second is found towards the middle.

After a period of several months the users job roll changes slightly, and the user finds him/herself using the second Web page more than the first. Slowly as time continues, the information score associated with the first Web page is reduced, and that associated with the second is increased. As the bookmark is re-ordered by user importance score, this leads to the first bookmark to slowly start descending through the bookmark file, while the second URL starts to ascend. After another couple of weeks, the second URL moves to the top of the bookmark file, whilst the second middles out. A couple of months later, the first book marked URL is completely removed from the Web, and thus the user no longer is able to visit it at all. In this situation the system quickly ascertains that the bookmark is no longer in use, so is rapidly moved to the bottom of the bookmark file for deletion. From the description given, it can be seen that the system can work in three possible modes. The first mode would be complete manual organisation, where the user simply rates each of the pages, and that order is maintained. The second mode would be semi-automatic, where the user and the system work co-operatively to maintain the book mark file. Here the user would initially rate a resource, and over time, the system would decide whether or not that rating was justified, modifying the 'user importance' score in a positive or negative direction as appropriate (N.B. the user has overriding control at all times). The third and final mode would be completely automatic. This mode would be entered if the user does not rate any of the URLs within the bookmark file. Here the system would completely automatically amend the user-importance scores and hence the book mark structure, organising the book mark file solely by what it judged to be most important.

Design Rationale

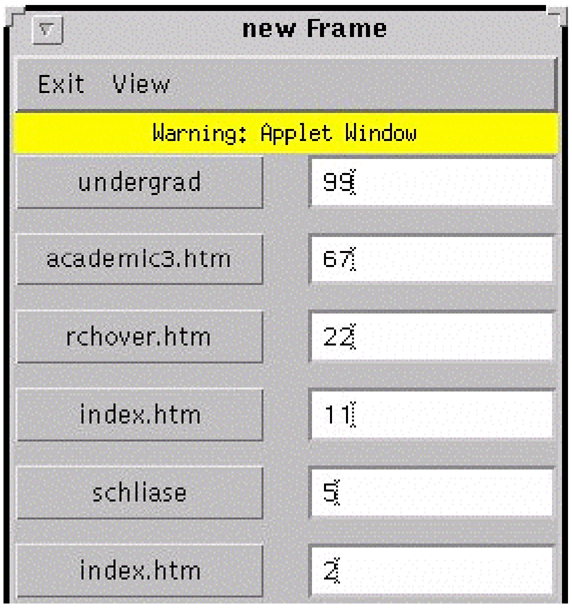
With the basic rules set down, a prototype was constructed to test their validity. The prototype was constructed in such a way as to put a minimum burden on the user, in terms of ease of use, ease of installation, and user interference. In order to fulfil these requirements many difficult design decisions had to be made, that all turned out to be heavily interwoven. The first and perhaps most important design decision taken however was, application, applet, or plug-in? The most important factor when designing the system was user acceptance. To achieve this aim, the system had to fit in, or appear to be a part of what the user already is familiar with. In this context, this meant that the system must run within the users actual Web browsing environment. The next most important factor was ease of installation. Even if the software were incredibly easy to use, no one would use it if it were too complex to install.

This left three options. The first and most complex option was to add extra functionality to the Web browsing software itself. Until recently this had been completely ruled out, as it has only been possible to get the source code for the NSCA Mosaic browser, that is seen as technically obsolete. However recently Netscape has published the source code for its popular Navigator Browser, allowing external developers to bolt on additional functionality (www.mozilla.org). The second option was to build a Netscape / Explorer plug-in. A plug-in is a piece of software designed to seamlessly integrate into a Netscape Web Browser. Again this is quite a complex solution to the problem, but at least the plug-in can be seen in its own right, without having to fully understand the intimate workings of the browser. The third option that was considered was to build the system as a Java applet run within the Web browser. Although this would not technically be as integrated into the browsing environment as the other two possibilities, with careful design and programming the applet could be made to appear as part of the Web browser.

After evaluating the alternatives, the application, and plug-in were soon dropped. The Web is built upon an IP backbone. The IP protocol has been designed with platform independence in mind, thus allowing all computers with a suitable implementation to talk to each other regardless of machine type, or operating system used. Using this technology, the Web has been built using a mismatch of hardware, all running different operating systems. The multi-platform nature of the Web suggests that any software used along side it must be able to seamlessly run across many platforms. On top of this, installing and upgrading Web browsers and plug-ins is very time consuming, and can be very complex, for example upgrading to a new Web browser, whilst retaining all of your plug-ins can be a hazardous affair to even the most computer literate people. Whilst upgrading a customised Web browser, would be fraught with difficulty. Barring these complexities in mind, the most rational choice seemed to be to build a Java applet, which would be completely platform independent, and require no-instillation, or complex upgrades. All that would have to be done is to publish the applet on a set Web page, and anyone could use it.

Interface

All of the preparation work led to a system with an interface (See Figure 2). Here the bookmark file is implemented as an ordered list of URL buttons. The interface is implemented as a separate frame, which opens outside the main browser window. When the user clicks on a URL button, that URL is loaded in the main browser window, and usage statistics are updated. The user rates the Web page though the score box next to the URL button. The user is able to override the current score at any given time, to manually move the bookmark within the file. It is this functionality which can enable the system to run in a semi-automatic mode.



undergrad	99
academic3.htm	67
rchover.htm	22
index.htm	11
schliase	5
index.htm	2

Figure 2: The Java Bookmark System bookmark list.

In order to re-order the bookmark, along with the users perception of how important a URL was several other metrics were recorded and stored in the database. These metrics included the number of times that the page had been accessed, the time at which the last page was accessed, and the total time that had been spent browsing the URL. The metrics were used to assign a score to each of the URLs within the bookmark. When the bookmark was to be refreshed, the standard deviation of all of the scores was calculated. Then the score associated with each URL was passed through a z scores algorithm, which calculated how far away the score was from the standard deviation (in both a positive or negative direction). The result of this calculation was added to the user ratings score, which were re-ordered using a straight insertion sort. Once the database had been re-arranged, then a new bookmark Web page was created, and the GUI re-drawn.

Architectural issues

We have already limited ourselves to constructing the system as an applet in Java, so what are the options? Well in fact there are only two, both of which require distributed computing. The first option would be to program the system using a Web server and CGI script written in Perl. Using the Java programming language it is possible to program an interface to a CGI system. Using this interface messages could be sent from the Client to a Web server, where they are processed by the Perl script. Whilst this solution would allow you to write a program with the necessary functionality, it would be hard to program the necessary level of user interaction. The second option would be to write a true Client / Server application. The Client could be written in any Web language such as Active X or Java, whilst the Server could be written in any language at all, which supported sockets.

The Java Bookmark system was designed using a client server architecture, with both the client and server written in Java. One of the most pressing arguments for the Client Server architecture was minimisation of impact upon the user. Although maintenance of bookmark files is a time consuming process, it is not processor intensive. Even when using modest hardware, the user should be able to add to or amend their bookmark files without noticing any interruption to other ongoing computing services currently executing. For the Java Bookmark system to be excepted by current users of Web browsers, the new software will have to offer a minimum performance overhead over that of the traditional bookmark. Even with great care taken to ensure implementation efficiency, it is very likely that performance will be impacted. It is therefore necessary to minimise this impact on the users Web browsing experience.

The client server architecture offers a novel way to help solve this problem by running processor intensive tasks on a remote server, which has been specially optimised for that purpose. Thus in principle the division of labour between client and server is such that the client is used merely for user interface tasks, and information flow control. While the server carries out the bulk of the processing. This process can be made all the more efficient by using the Java RMI architecture, which allows procedures to be called on the remote server from the client.

In this case the client program was responsible for building the GUI from a standard Web page; displaying the URL to be viewed in the main browser window, and opening an IP connection to the server, such that the browsing information could be saved before the system shut down. It is interesting to note however that the client only required the functionality to send data to the server. This was because the client was constructed in such a way as to build its interface from parameters written to a Web page. When the interface was to be refreshed, the server simply replaced the current Client Web page, and the GUI was redrawn. In a way this meant that the software was upgraded each time it was refreshed, and goes to show how easy upgrading the whole of the software could be. On the other hand the server was responsible for decoding messages from the client; saving and querying data from the database; re-ordering the bookmarks, and generating the standard Web page from which the bookmark GUI was created.

Another more subtle reason for using the Client Server architecture was brought about by the security model that prevents applets from accessing local computing resources. Applets written using the 1.02 and 1.1.x implementations of the Java programming language are subject to stringent security restrictions. The most notable of these is that the applet is not allowed to read, or write to the local file system; and can only make an IP connection to the Web server that it was downloaded from. These restrictions make programming complex applets troublesome, but are there to prevent rogue applets reading from or writing to the file system, or making uncontrolled IP connections. It is obvious that the book mark system that has been described will need to have some kind of persistence. The only way to provide this persistence is to write a server as a stand-alone application running on the machine where the applet resides. This server can then open up other IP connections under the control of the client to access any other resource via the TCP/IP protocol. To provide the necessary persistence, the intelligent book marking system used a Java server to connect to an Oracle Database via a JDBC interface. The JDBC interface is a Java API coupled with database specific drivers that allows a Java program to directly connect to a database, and update or retrieve information by passing simple SQL. As the database interface and commands were executed via the server (a Java application) this meant that the database could reside on a remote machine from the Web Server. Figure 3 illustrate the three-tier architecture.

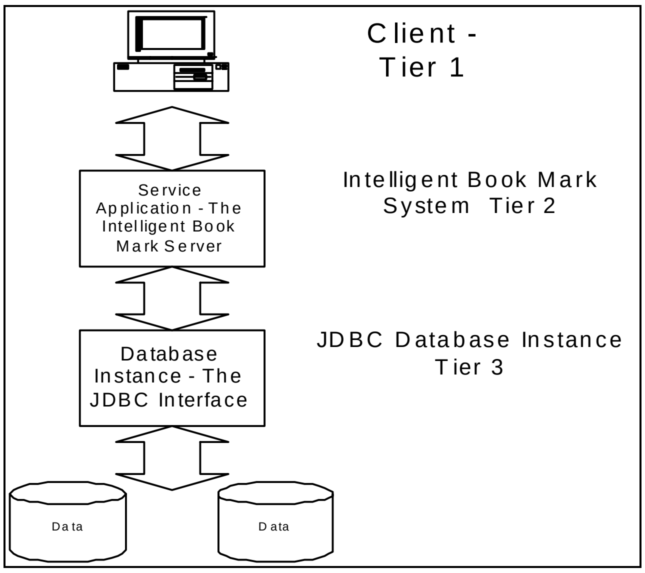

Client - Tier 1
Service Application - The Intelligent Book Mark Server
Intelligent Book Mark System Tier 2
Database Instance - The JDBC Interface
JDBC Database Instance Tier 3
Data
Data

Figure 3: The Java Bookmark System architecture

One of the main advantages of the intelligent bookmark system is that it requires no user installation. The server would be set up on the Web by a qualified technician, so that all the user will have to do to get the system working is to locate the desired Web page. As all the user interfaces are applets, they are automatically downloaded and instantiated. The other main advantage to this is that the system can easily be upgraded. Traditionally upgrading software is a very expensive and time-consuming task. Here all that needs to be done to upgrade everyone’s system is to update the software on the central server. Next time the user visits the Web page, the latest version will automatically be downloaded. The main constraint imposed on the user is that the server is proprietary. It would be difficult for a user to export information from the server, or link (collaborate) the activities of this server, with any other server that they are using. The main reason for this constraint, is the same as with many systems, in that the system was programmed with its own functionality in mind, rather than connectivity. However, if many users used the system then a central pool of bookmark data would be created. This data could be used to pool resources for future research.

4. The BASE Framework

What were the basic choices we were faced with in the design of the semi-automatic bookmark management system? Inspiration could be drawn from state-of-the-art computer science frameworks characterising computer application architectures. However, this would lead to emphasising technical issues only, and not issues related to the installation and use of the technology. Therefore, the following suggest a framework characterising the basic choices for providing Web navigation support, based on recognition of issues concerning the installation and use of the technology. The framework has emerged from considerations when designing the prototype. The BASE framework (see Table 1) classifies Web navigation support according to how it is provided, i.e., support built into the Web browser, a separate application, a server facility, or embedded into a Web page. In the following each of the four categories are discussed in detail. The four categories represent an analytical distinction of possible components since many systems can be characterised as a combination of the four types of elements.

BASE Web Navigation Architecture Components	
Browser	Part of or the entire Web navigation support functionality is provided as an integral part of the Web browser
Application	Part of or the entire functionality is provided in a separate application which is installed on the client computer
Server	Part of or the entire functionality resides in a server. Access to this functionality can either be through a Web page or by a client application
Embedded	Part of or the entire functionality is embedded in or part of a Web page downloaded, such as browser plug-in, JavaScript panel, or Java applet

Table 1: The BASE Web navigation architecture framework.

Browser

The Web browser, here interpreted as the code showing the contents of html pages, can provide navigation support. Current browsers have rudimentary navigation support such as the bookmark, and history logging facilities, backwards and forwards arrows, auto-complete URL, and preference settings for search engine (Sørensen, 1998). However, even more complex navigation support could be built into the browser. The release of the source code for Netscape, browser additions could be particularly feasible (Charles, 1998). A modified browser targeted a certain community, such as stockbrokers, journalists, marketing people, could support the search for new Web resources of interest to this community and provide partly automatic pruning of bookmarks based on domain specific filters. The advantage of this approach could be relatively easy installation and use because the navigation support can be completely integrated into the browser. Installing the functionality would not be any different from installing a browser, and using it could be similar to using an ordinary browser. On the other hand if users do not change browser frequently, this approach could have difficulties in being accepted.

Application

The navigation support functionality can also be provided as a stand-alone dedicated application such as the bookmark management applications available. Here, the advantage is that the user directly can identify the adopted solution to the perceived problem of navigation the Web since functionality clearly signals its existence through the entity and identity of the application. It can be viewed as a "software gadget" addressing the navigation problem. A possible disadvantage with this approach is, however, the potential demands of the user with respect to installing and maintaining the application. Since it is not entirely integrated with the Web browser, issues of compatibility may occur. Updating the operating system and the Web browser can eventually lead to the need for upgrading and installing new versions of the application. The application can provide basic support for recording the navigation history and for managing and annotating bookmark files. It can also provide Web agent semi-automatic functionality for accessing and downloading Web resources.

Server

The Web is primarily a client-server architecture, with the Browser clients accessing Web servers (Huhns, 1999). A server facility can provide navigation support which otherwise can be difficult or impossible to obtain. This can, for example, be in the form of filtering operations that are computationally taxing and which rely on large thesauri. One of the advantages of a server-based solution can be relative easy installation and use of the functionality. The user may not need to install special software but can interacts with the server through Web pages or Java Applets that automatically are downloaded to the client computer from the server. If a large community of users use a serer-based service, the feedback they provide can enhance the server's filtering capability (Oard, 1997). Maintaining the service will not be the responsibility of the individual user, but is instead left to the providers of the service. This may still cause problems for the user, since using the facility may require of the user to register by filling in forms. The user may also be required to log on to the server each time, thus requiring of the user either to set the Web browser to allow cookies with password to be stored, or alternatively to remember the user name and password. This may seem trivial to accomplish, but with an increasing number of user names and passwords, the management of these becomes a task in itself. Furthermore, because most of the services available are offered on a global market, it is unlikely that most users will be able to choose a user name that is easy to remember. A server can provide both complex processing, recommendations based on other users, and ease of installation and maintenance. However, in terms of performance, the server will be a bottleneck slowing down the navigation process if it records the behaviour of a large number of people. Obviously, increasing bandwidth and the possibility for setting up mirror sites can alleviate this problem.

Embedded

The Web navigation functionality can also be implemented as embedded functionality in terms of executable contents in Web pages, or as plug-in extensions to the Web browser. It can be JavaScripts embedded in Web pages, or Visual Basic, Java or ActiveX components that are downloaded automatically to the client. Extending the Web browser by means of a plug-in offers the opportunity to provide ubiquitous support. Once installed, the plug-in will be ready for use whenever needed. However, the disadvantage of the plug-in is the overhead needed for installing and updating the plug-in. Maintaining the functionality will require detailed knowledge of where the Web browser plug-ins are located. Ensuring compatibility between evolution of browser, plug-in and operating system can cause practical problems. Other solutions, such as Java applets or JavaScripts offer the advantage of automatic download and installation of the required functionality with the price of the functionality not necessarily being available instantly.

5. Assessing Technologies

How can we then apply the BASE framework in order to characterise the architectures of Web navigation support and to discuss the installation and use of the technology? The combination of the Web navigation and the BASE framework, for example, provides a map of possible options. Figure 4 shows the mapping of a number of Web navigation support systems onto the choice of architecture.

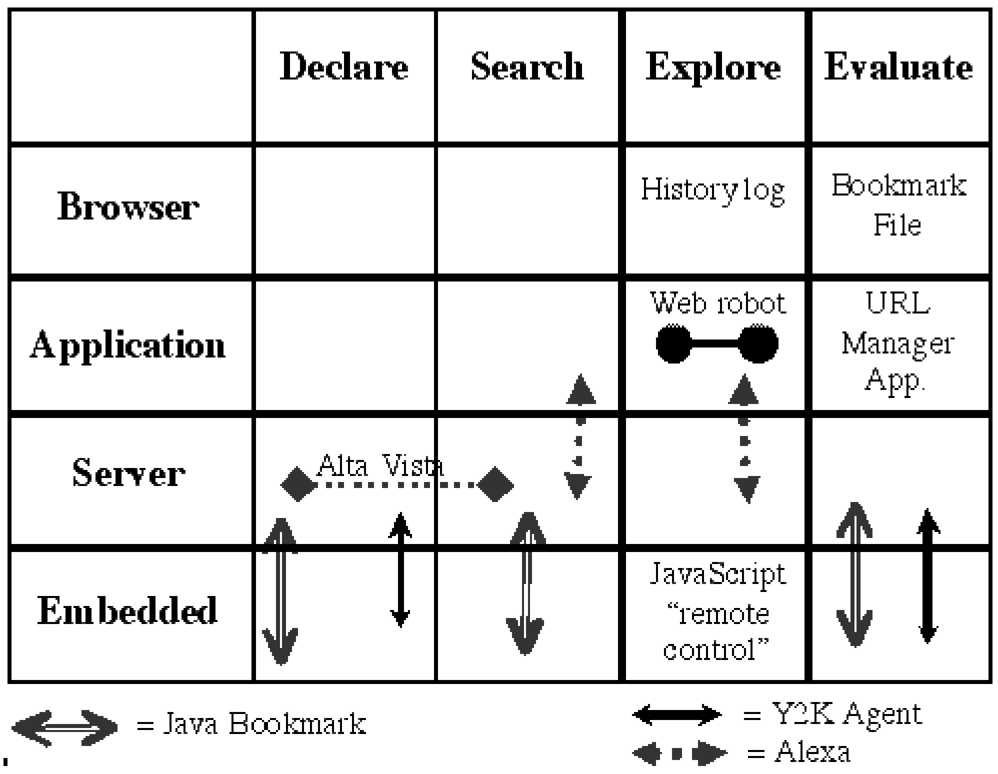


Figure 4: Mapping Web navigation activities supported onto the BASE framework. Example systems are chosen to illustrate that some systems apply several components.

The browser history log represents an automatic trace of the user exploring Web resources. The bookmark file is the result of a manually updates list built as a result of the user evaluating Web resources. Setting the browser preferences for a particular search engine supports easy integration between the browser and search activities.

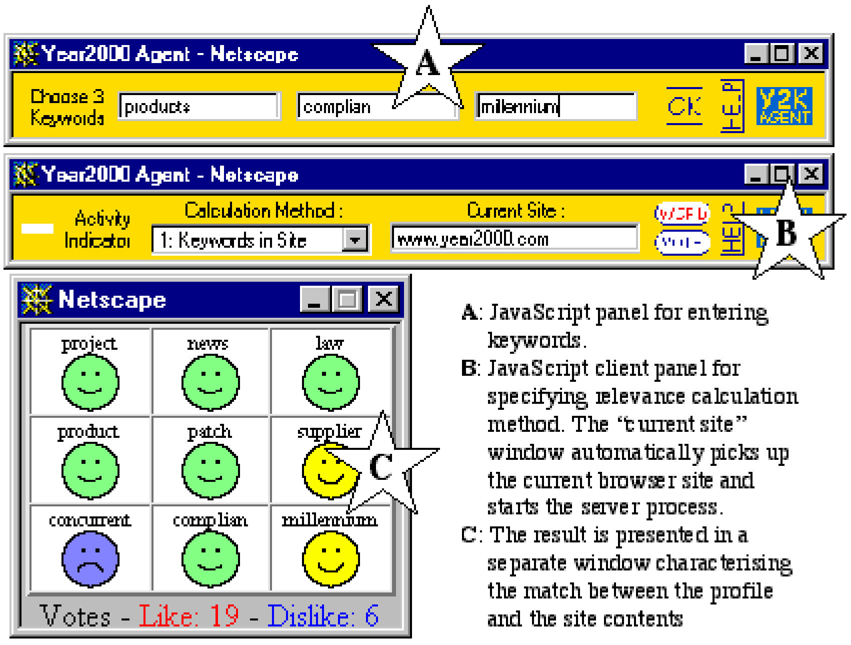
A host of URL manager applications are available. They generally support evaluation of Web resources by providing a user-friendly interface to support complex bookmark management activities. They support the user in maintaining a proprietary database of Web resources across different Web browsers. Some of these applications will also support automatic recording of explored Web resources.

The Web robots available generally allow the user to specify a set of URL's to be visited and downloaded. This supports the exploration of Web resources. Some of the systems are either based on client-side proxy-servers containing either previously visited Web pages or on download of Web pages anticipated based on links on pages downloaded. Some of the more advanced applications (such as BullsEye, www.intelliseek.com) support filtering of the results, i.e., and example of semi-automatic evaluation based on a profile. Search engines such as Alta Vista and Hotbot, sends out many robots indexing the Web, and provides server-side access to the index.

Alexa (www.alexa.com) is a stand-alone application that sits in an external window to the Web browser. The application interacts with a central server. Alexa supports search and exploration of Web resources by tracking behaviour of all Alexa users. As a given Web site, Alexa will provide information of the typical Web sites others visit after having been at the current site. As such it functions as a mutual recommender system implemented as a client-server architecture where the client is a stand-alone application. Alexa, thus, supports the search and exploration of web sites through a client-server architecture.

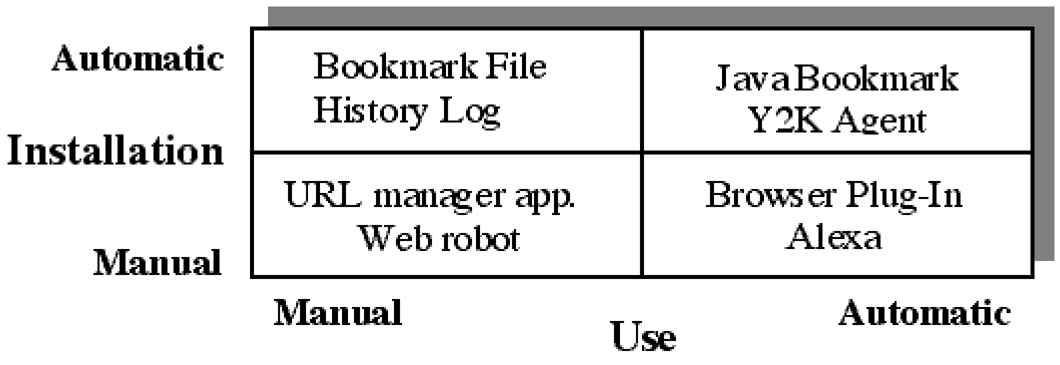
Increasingly Web sites have built-in JavaScript control panels supporting interaction with the site. Providing an instantly available index of the site supports the user in exploring the contents. The Java Bookmark system described previously, is an example of a system supporting declaration, search and evaluation of Web resources. It does not as such support evaluation, since the user is required to visit Web pages, but the combination of observed behaviour and explicit rating are used as parameters in the automatic ranking of bookmarks according to perceived relevance. The Y2K agent developed by Trowbridge (1999) (seen Figure 5) is an example of an architecture similar to the Java Bookmark system. It is a client-server architecture where the server on regular intervals indexes Web sites about the Year-2000 problem. The JavaScript client control panels (see Figure 5) are embedded access points to the server database. The user can specify keywords, and based on these, the Web site is rated according to the relevance. This client-server system supports the user in declaring an interest profile that is used as a filter supporting the process of evaluating the potential relevance of the Web site regarding the particular interest. The strength of combining a server-side index process by means of a Web robot with the JavaScript client is similar to the Java Bookmark System that the end-user will be presented with a service with no additional overhead for installation. The responsibility for maintaining the server process will reside with the organisation providing the service.

Characterising technologies according to the process of installing and using them (see Figure 6), we see that the URL manager application or a Web navigation robot requires manual installation and will also require manual activities in use situations. The user may have more sophisticated functionality available in the URL manager application, but the user is still required to do most of the work maintaining the bookmarks. The typical Web robot is at most semi-automatic in the sense that it will traverse the proportion of the Web specified in a list of URLs or by automatically downloading the links on the current Web page being displayed by the user's Web browser.



A: JavaScript panel for entering keywords.
B: JavaScript client panel for specifying relevance calculation method. The "current site" window automatically picks up the current browser site and starts the server process.
C: The result is presented in a separate window characterising the match between the profile and the site contents

Figure 5: The Y2K Agent (Trowbridge, 1999), supporting declaration and evaluation of pre-indexed Year-2000 Web sites.



Installation \ Use	Manual	Automatic
Automatic	Bookmark File History Log	Java Bookmark Y2K Agent
Manual	URL manager app. Web robot	Browser Plug-In Alexa

Figure 6. Classification of navigation support systems based on distinction between the installation and the use of the technology from the perspective of the end-user with a Web browser.

The bookmark file, the history log and the URL auto completion functions provided by standard Web browsers are automatically installed when the Web browser is installed. It is here assumed that given the Web browser is by far the most common method of accessing the Web, it does not yet make sense to treat the browser as an optional mode of access. The Web browser *de facto* defines the Web for most end-users. Using the features built-into the Web browser requires of the user to manually conduct the navigation.

Navigation support provided through a browser plug-in such as CyberViewer (http://www.easystreet.com/~jasonk/cyberviewer/) which automatically keeps a list of thumb-nail pictures of pages visited, or Alexa (www.alexa.com), discussed above, both have to be downloaded and installed by the user. However, once this task has been completed, they will both provide the navigation support automatically. The Java Bookmark System and the Y2K Agent are both installed automatically when the user selects the server homepage. The uses of the two systems are automatic in the sense that they automatically link the Web browser to the server. However, both systems allow the user to perform additional configuration in order to optimise performance. Increasing use of JavaScript control panels that interact with servers and which, based on a user-profile can filter and present interesting links and summarise information, seems to be one of the viable strategies for developing navigation support which is both powerful as well as easy to install and use.

Recent developments increasingly tie the operating system, the Web browser and Web search together. The recent court case between Microsoft and the US Department of Justice concerns amongst others the integration of the Internet Explorer browser and the Windows98 operating system. Similarly, the Sherlock search function on MacOS 8.5 from Apple integrates searching local disks and searching the Web. Each Web site needs to produce a small plug-in specifying the site address and protocol. The protocol is downloaded and becomes part of the operating system.

6. Conclusion

As argued by Nielsen, one of the main challenges facing the Web is the development of a huge amount of Web sites by people who are not experts in developing user friendly technology (Nielsen, 1999). Another, and subsequent, challenge is to provide the necessary support for navigating all these Web sites once they are built. Attempting to understand and develop the possibilities for supporting Web navigation is in some respect a hopeless task. The incredible growth of the Web has spurred a global race for developing advanced support. At the same time, most of the support developed is far from penetrating the market. Both consumers and professionals will in general use the basic bookmark management functionality built into the browser they are using. They are also likely to use services provided by site categorisation sites such as Yahoo.com or search engines such as Hotbot.com. The more substantial technology supporting user modelling, information filtering, mutual recommender functionality, neural nets etc, are not mature enough to reach the mass market. In order to facilitate this innovation process we must understand the basic choices for supporting Web navigation, and we must constantly bear in mind that the customer no longer is the expert computer scientist but consumers and professionals.

This paper has taken a first small step in this direction by proposing the BASE framework, which outlines the technological choices for Web navigation support. This is, however, just a first step, and one that needs to be tested in a setting substantially different from the laboratory in which it was conceived. To validate the BASE framework suggested in this paper, we must carefully study how the categories map onto categories of navigation support elicited from real life information filtering and retrieval situations. Given the argument we make is based on the perspective of optimising both installation and use of web navigation support technologies, the next step obviously must be to relate the framework to a real setting. An obvious first point is to study how the BASE framework relate to information retrieval situations in knowledge intensive situations. The framework could also be theoretically strengthened by relating it more precisely to emergent technologies. As a means of classifying new Web navigation technologies, important feed-back could be provided regarding the general validity of the framework.

It is also a future challenge to study how we can employ complex navigation technology, such as the ones discussed in this paper, in actual work settings.

References

Allen, R. B. (1990): User models: theory, method, and practice. *International Journal of Man-machine Studies*, vol. 32, no. 5, pp. 511-543.

Balabanovic, M. and Y. Shoham (1997): Fab: Contents-Based, Collaborative Recommendation. *Communications of the ACM*, vol. 40, no. 3, pp. 66-72.

Belkin, N. J. and W. B. Croft (1992): Information Filtering and Information Retrieval: Two Sides of the Same Coin. *Communications of the ACM*, vol. 35, no. 12, pp. 29-38.

Charles, J. (1998): Open Source: Netscape Pops the Hood. *IEEE Software*, vol. 15, no. 4, pp. 79-81.

Darken, R. (1998): Breaking the Mosaic Mold. *IEEE Internet Computing*, vol. 2, no. 3, pp. 97-99.

Etzioni, O. (1996): The World-Wide Web: Quagmire or Goldmine? *Communications of the ACM*, vol. 39, no. 11, pp. 65-68.

Fagrell, H. and P. Ljungstrand (1998): Make and Agent and You Shall Find. In *21st Information systems Research seminar In Scandinavian, August 8-11, Sæby Søbad, Denmark*, ed. Peter-Axel Nielsen, Niels Jacob Buch, and Lars Bo Eriksen. Aalborg University, pp. 197–206.

Huhns, M. N. (1999): Networking Embedded Agents. *IEEE Internet Computing*, vol. 3, no. 1, pp. 91-93.

Joshi, A. and M. P. Singh (1999): Multiagent Systems on the Net. *Communications of the ACM*, vol. 42, no. 3, pp. 39-40.

Krulwich, B. (1997): Automating the Internet: Agents as User Surrogates. *IEEE Internet Computing*, vol. 1, no. 5, pp. 34-38.

Lieberman, H. (1995): Letizia: An Agent That Assists Web Browsing. In *International Joint Conference on Artificial Intelligence, August, Montreal*.

Lyytinen, K. and S. Goodman (1999): Finland: The Unknown Soldier on the IT Front. *Communications of the ACM*, vol. 42, no. 3, pp. 13-17.

Maes, P. (1994): Agents that reduce work and information overload. *Communications of the ACM*, vol. 37, no. 7, pp. 31-40.

Maglio, P. P. and R. Barrett (1997): How to Build Modeling Agents to Support Web Searchers. In *User Modelling: Proceedings of the Sixth International Conference, UM97, Vienna*, ed. Anthony Jameson, Cécile Paris, and Carlo Tasso. Springer Verlag, pp. 5-16.

Nielsen, J. (1999): User Interface Directions for the Web. *Communications of the ACM*, vol. 42, no. 1, pp. 65–72.

Oard, D. W. (1997): The State of the Art in Text Filtering. *User Modeling and User-Adapted Interaction: An International Journal*, vol. 7, no. 3, pp. 141-178.

Schneiderman, B. and P. Maes (1997): Direct manipulation vs Software Agents: Excerpts from debates at IUI 97 and CHI 97. *Interactions*, no. November-December, pp. 42-61.

Schubert, C., R. Zarnekow, and W. Brenner (1998): A Methodology for Classifying Intelligent Software Agents. In *ECIS98, Aix-en-Provence*, ed. Walter R.J. Bates, vol. I, pp. 304-316.

Sørensen, C. (1998): Where Have You Been Today? Investigating Web Navigation Support. In *21st Information systems Research seminar In Scandinavia, August 8-11, Sæby Søbad, Denmark,*, ed. Peter-Axel Nielsen, Niels Jacob Buch, and Lars Bo Eriksen. Aalborg University.

Tauscher, L. and S. Greenberg (1997a): How People Revisit Web Pages: Empirical Findings and Implications for the Design of History Systems. *Journal of Human Computer Studies*.

Tauscher, L. and S. Greenberg (1997b): Revisitation Patterns in World Wide Web Navigation. In *ACM SIGCHI '97 Proceedings of the Conference on Human Factors in Computing Systems, Atlanta, Georgia*. ACM Press.

Terven, L., W. Hill, B. Amento, D. McDonald, and J. Creter (1997): PHOAKS: A System for Sharing Recommendations. *Communications of the ACM*, vol. 40, no. 3, pp. 59-62.

Trowbridge, M. (1999): *Y2K Agent*. Thesis Computer Science, Aston University.