1996

On Mobile Systems and Disconnected Browsing of Distributed Information

Anupam Joshi

Sanjiva Weerawarana

Elias N. Houstis
Purdue University, enh@cs.purdue.edu

Report Number:
96-040

ON MOBILE SYSTEMS AND DISCONNECTED BROWSING OF DISTRIBUTED INFORMATION

Anupam Joshi
Sanjiva Weerawarana
Elias N. Houstis

Department of Computer Science
Purdue University
West Lafayette. IN 47907

CSD TR-96-040
June 1996

On Mobile Systems and Disconnected Browsing of Distributed Information*

Anupam Joshi, Sanjiva Weerawarana, and Elias N. Houstis
Department of Computer Sciences
Purdue University
West Lafayette, IN 47907-1398

E-mail: {joshi,saw,enh}@cs.purdue.edu

Abstract

The software and protocols associated with the World Wide Web are designed with static hosts in mind. Static hosts are connected to wired, high bandwidth networks, and are capable of transmitting and receiving large amounts of data without significant delays. Therefore the size of the incoming data files has never been a concern. However, this causes problems when information access is desired on mobile hosts (MH), since data transmission over a wireless network is much slower than on a wired network. Mobile computers are also relatively resource-poor, compared to their desktop counterparts. This fact is ignored by HTTP servers, and large data files are transmitted to computers that cannot properly display them. Also, mobile computers operate in constantly changing network environments. It is possible for a mobile computer to become temporarily disconnected from a network when it changes base stations or goes out of range of a base station. A mobile host may also *doze off* to preserve battery power and thus be disconnected. If at the time it goes down, a mobile computer is communicating with another computer (mobile or static), it should be able to tolerate the fault of temporary disconnection. This work focuses on addressing these problems in the context of web browsing from a mobile host. The current model of Web browsing is inherently sequential, and wasteful of bandwidth. This paper investigates an efficient model for browsing and describes the design of a smart Web browsing application which performs transactions based on the user's available resources and manages disconnection.

1 Introduction

Recent years have seen a significant increase in interest generated by mobile computing. With the advent of the World Wide Web[4], browsing (surfing) the web is becoming an increasingly common activity for computer users. The current model of browsing the web, however, is sequential, and one that leaves most of the burden of finding the relevant information on the user and his computer.

*This work was supported in part by NSF awards ASC 9404859 and CCR 9202536, ARPA ARO award DAAH04-04-G-0010 and a grant from the Intel Corporation.

In other words, a user is expected to know starting points, and then search through documents and follow links to find the information s/he needs. Clearly, this means that while the user is searching for information, a network connection needs to be constantly maintained. Moreover, a lot of (potentially) useless information is transmitted over the (wireless) network, wasting precious bandwidth. This mode of information access is clearly not suited for web browsing from a mobile platform. Mobile platforms, for one, are connected over wireless links. Even the best of such networks provides far lower bandwidth than run of the mill wired networks. Wireless LANs typically provide a total bandwidth of 2Mbps, and throughputs in the hundreds of Kbps range. Wide area wireless networks (mostly cellular) operate at tens of Kbps. Moreover, unlike wired networks, disconnections are a frequent phenomenon in wireless networks. These occur while a host is between base-stations, or when it falls in a "radio-shadow" area. Disconnections in the mobile environment can also be elective. In other words, the mobile host may chose to "switch off" certain part(s) of its functionality in order to preserve battery power. We refer the reader to [9] for details of the mobile computing scenario. A related problem is the capability of the mobile platform. Web servers do not take into consideration the network connection between them and their clients. Rather, the server merely returns whatever document was asked for. Furthermore, users are routinely creating multimedia-rich pages. In fact, so heavily is multimedia used that often "text only" versions of these pages have to be made available for users accessing them across WANs. Wireless networks do not have the data transmission speeds of wired networks, so downloading large files over wireless networks, even local area ones, consumes a longer time. Not only does it take longer to download large files, but more battery power is consumed, reducing the time the mobile computer can be used. Web servers also have no notion of the resources available at the client's end, and blindly assume that the client is capable of properly displaying the data that it receives. For example, a user may follow a hyperlink to a sound file on an X terminal which is not capable of playing sound. The server will blindly supply this data to the client, consuming time and network bandwidth to deliver a sound file that cannot be used. Such problems are especially acute on mobile computers, which are not as resource-rich as their desktop counterparts. Text only pages, while helpful in this context, force and unnecessary all or none choice on the user.

While we have articulated these problems in the context of web, they generally relate to information access from mobile platforms to distributed data. The Internet protocols are primarily designed for static networks where changes are rare and disconnections catastrophic. HTTP is a typical such protocol, and has no knowledge about the state of the system, including the underlying network. Note that the proposed HTTP-NG[5], presently being discussed, will have some notion of the context of the request. The attempt in this work is to add some intelligence to the present http client server system in a manner that will be interoperable with HTTP-NG. Providing a reasonable level of performance in the face of frequent disconnections, restricted bandwidth and limited resources is a major issue in mobile computing. In context of information access, we refer to this problem as *disconnected browsing.* Note that our use of this term is in a much broader sense than in some commercial products. In these products, the only concern is to be able to operate in a mode where predictable disconnections from the network occur in an otherwise high bandwidth connection. In this paper, we describe a software architecture to support disconnected browsing, and present preliminary implementations of two component systems called *Mowser* and *WebIQ.*

1.1 Some Application Scenarios

Before going further, let us consider some scenarios where disconnected browsing is useful. These will also serve to illustrate the potential commercial impact of research in this area.

1.1.1 Military

Tactical Picture is the totality of the information needed by a military commander. Accurate, up to date, and relevant information can allow commanders to make optimal decisions. In a military scenario which is becoming increasingly information centered and "smart", it is perhaps belaboring the obvious to point out that the better the tactical picture, the better is the ability of the commander to conduct operations.

While easy to state, accomplishing this requires significant improvements in the state–of–the–art in various computer science research areas. The primary reasons for this are threefold. First, there is an ever increasing amount of data becoming available on–line. All of this data could be potentially relevant to the situation, and it would be incorrect to assume that only certain sites (military data repositories) need to be searched. For instance, a naval commander in a vessel off the coast of the Former Soviet Union might face a situation where his decisions could be aided by the latest wire reports being put out by news agencies such as Novosti or Tass or InterFax. The second major obstacle lies in fact that the answer to the commanders query could require agglomeration of information from diverse sources. For example, it could be information about troop movements from satellite and information about an attempted coup from CNN that might together enable a commander to appropriately judge the situation. Finally, the commander (or the soldier) in the field is likely to be connected over a low bandwidth wireless link, which may be subject to enemy jamming. Thus the information access process must not rely on a disturbance free, high bandwidth connection.

Clearly, with the plethora of ever changing information, expecting or requiring a commander to be aware of all possible sources that might contain information of relevance is absurd. Techniques that will automate the process, and seamlessly allow the commander to access information as well as computational resources are required. This integration is extremely important, since generating the tactical picture may require not just passive gathering and display of information, but also require that some computations be done based on this information.

1.1.2 Medical

In emergency and trauma situations outside the hospital, accurate patient information and medical history can play an important role. Consider an accident that occurs on the interstate, several miles from the nearest medical facility. Given the identity of the patient, disconnected browsing techniques could retrieve information and pass them onto the EMS workers' palmtop machines over low bandwidth wireless networks. The information traffic would be two way – sending the information on the patient's condition (wounds, blood pressure, pulse etc.) to the doctors in the hospital where s/he will be sent.

2 Related Work

There is little existing work in the area of web browsing from mobile platforms. The two mutually distinct areas of related research are offline browsing and dynamic browsing. The first is the perusal of information available on the WWW, while being disconnected from the network. Most current work in this area involves downloading entire documents to the mobile platform and caching them for later browsing. We believe this approach does not suit a resource-poor platform as it poses additional burdens in the form of larger disk requirements. Dynamic browsing, on the other hand involves integrating location information to the query being sent. This information is used to retrieve information relevant to the user's current location. While these projects suggest interesting uses for the WWW, they are based on additions to the current protocols, and do not address the problem of viewing the information that is currently available on the WWW, from a resource-poor mobile platform that is frequently disconnected.

Several offline browsers are commercially available. WebWhacker[26] , and Grab-a-Site[1] are offline browsers that download documents into a local disk for later viewing. The user specifies a set of links, similar to a bookmark file, which the application proceeds to download and cache. As hyperlinks are encountered in the downloaded documents, these are also transferred to the local machine, creating a mirror of the given sites. They allow some level of control over what is being downloaded, by limiting the number of links traversed for each URL, or the total disk space used. Such utilities generally edit the html files that are downloaded and replace their links to point to the cached files.

WebWatch[25] is a similar product, that in addition, periodically looks for changes on the cached sites. An agent-based approach is used to periodically query the sites for updated documents and refresh the local cache. Milktruck[17], another offline browser, runs a proxy web server on the portable machine to deliver the cached documents. After downloading the requested documents, the user sets his/her browser to point to the proxy server run on the mobile host. The proxy server then serves documents from its local cache, without using a network connection. Milktruck also offers the facility to refresh the cache periodically by querying the original server.

The common feature of these products is that they propose to conserve one scarce resource (connection bandwidth) by using up another scarce resource (disk space). Furthermore, none of the mobile platform's resource limitations are considered in these systems. We believe this is a very narrow view of "disconnected operation".

Mobisaic[24] is a system that extends the current WWW protocols to encode location information in the URL. It uses a "dynamic environment" introduced by Xerox PARC[21] to deduce the mobile user's current location and create a "dynamic URL". Though this system can provide useful interaction with custom applications that understand location information, it does not provide any advantage in browsing the rest of the WWW. Dynamic documents[13] are another example of mobile access to the WWW. In this case, the burden of processing location dependent information is placed on the mobile computer, thereby using more of its resources. Another location-dependent information service is described in [2], with a description of a building navigator application. These applications propose extensions to the current WWW and its protocols, in order to deliver information relevant to a mobile platform (i.e. location information). However, they do not deal with the issues of disconnectivity, and resource-scarcity of mobile platforms.

Glomop[10] has an approach similar to the mowser component[12] of our system. Glomop operates a proxy server on a stationary machine, through which the mobile users make their requests. The proxy server retrieves the requested documents and forwards them to the mobile user, with the exception of images. Glomop scales the images down to a smaller size before transmitting them to the mobile user. This reduces the bandwidth requirement between the mobile platform and static host, and allows for faster transmission times.

Some prior work, in unrelated contexts, has been done on searching for information on the Internet. Locating specific information in the Internet is becoming more difficult due to the its explosive growth and diversity. Many search engines and research papers have attempted to address this problem. However, these works for the most view this as a pure information retrieval problem. The reader is likely familiar with several "crawler" based search engines such as Yahoo, AltaVista, Lycos, InfoSeek, etc. etc. In the following paragraphs we will describe some other important systems which have come out from the research community.

Oates et al. [19] indicate two distinct methods of searching in their paper on cooperative information gathering. The first is information retrieval (IR) systems which assume that the user has the knowledge to indicate what is to be searched for and the knowledge to filter through the results for quality documents that meet their expectations. Information gathering (IG) involves the pro-active acquisition of information. This may require analysis of intermediate results and a satisficing search.

SavvySearch[8] is a search engine that allows parallel searching on multiple information retrieval resources. The user is allowed to choose which information resources the search should be performed on. This type of architecture does little to alleviate the problem of searching the rapidly growing Internet. Information is provided from multiple sources, but the search is done with no attempt to categorize the information in order to develop an efficient method of searching. SavvySearch is the Goliath of brute force searching.

NEXOR's ALIWEB[16] requires a site to create a file, either manually or electronically, containing a description of their services in a standard format. The site then informs ALIWEB of the file. ALIWEB routinely retrieves the files and creates a searchable database of the services. This is a method of categorizing information, but it does not allow for any details about the quality of the services. The Harvest architecture[22] is similar to ALIWEB since a site creates an index of its resources. A gatherer then retrieves these indices and passes the information, after filtering, to brokers. A search is then conducted by contacting a broker who may have the required information or can contact other brokers if necessary. Like ALIWEB, Harvest lacks relevance feedback.

WebHound[15] is a service that attempts to solve the problem of relevance feedback by giving the articles that it supplies rankings based on the opinion of users that have read the articles. After supplying an article to a user, WebHound expects the user to read and rank the article on its quality. A user can request specific articles by supplying a minimum quality rating. WebHound is not a true search engine because the articles that it supplies are recommendations. Therefore, WebHound may not have information on many topics of interest.

DeBra and Post[6] suggest the use of the fish-search algorithm. The fish-search algorithm is an IG type of search. During a fish search, documents relevant to users request are searched for links to other documents. The search expands outward along the links found in documents from the initial search. The search will continue to grow by following links found in the new documents. DeBra and Post visualize this as a school of fish finding food (a relevant document) and producing children (new searches along links in the document). This method is efficient because searching links from a relevant document uses less resources than continual brute force searches and is more likely to provide relevant documents.

Another method of improving the efficient use of the Internet is proposed[20] by Pitkow and Recker's study of document access and individual users usage patterns. Their study shows that user patterns can be identified. After identifying these patterns, documents can prefetched or temporal patterns can be automated. A web search engine can be made more effective by combining the methods described above. For example, a user-profile could be built, as in Webhound. This could be used as the basis of a search using collaborative information gathering techniques similar to that of the Harvest system. Finally, if the search does not return successfully, a brute force search can be done, like in SavvySearch.

3 Software Architecture

Our proposed software architecture builds on the widely accepted model for mobile computing [3]. In essence, it assumes that the wired network is a high bandwidth pipe. Failures of either the network or host machines on this part of the infrastructure are rare and catastrophic events. The mobile hosts, on the other hand, can frequently disconnect, either electively to preserve resources, or otherwise due to a dropping of the wireless connection. The mobile hosts (MH) are supported by mobile support stations (MSSs) which act as gateways between the wired and wireless networks. Each MSS supports MHs within its cell, similar to how cellular phones are organized. MHs within a cell can directly communicate with each other, otherwise they must communicate through their MSS, which in turn talks to other hosts and MSSs on the wired network.

Our architecture adds a proxy server on each MSS to the basic MSS-MH model. This will function as the gateway to the Web for browsers running on MHs. All information requests will be routed through this proxy, which will operate in a manner transparent to the user. The MHs will be able to still run the user's favorite Web browser, as long as it is forms capable and allows the provision of a proxy mechanism. The proxy will implement the new model we are proposing for disconnected browsing. Essentially, the objective is to provide some state information which will enable us to handle the constraints imposed by mobility. In other words, we are adding some intelligence to the network, so that it adapts to changing resources available. A more general vision of such a proxy, suited to computational rather than information access tasks, can be found in our prior work[7]

Running a proxy server allows us to push all the intelligence into this unit, and also have it maintain needed state information. This allows us to use existing IP based systems as is, and allows retention of popular tools and software such as Web browsers and servers. In other words, legacy stateless protocols effectively become statefull. We illustrate this in Figure 1. For disconnected browsing, the proxy has two distinct but related functionalities, which we call Mowser[12] and WebIQ[14] respectively. Mowser performs what we call multimedia data transformation on the documents being served to the browser on the MH. For example, it will notice that the image being sent to an MH with an 8 bit color display has 24 bit data. It will then transform this image to 8 bit, and send this reduced data over. Along the same lines, even if the host had a 24 bit color screen, it would reduce the color and the size of the image so as to not make the user wait an inordinate time after the request due to the low bandwidth available. The functionality represented by WebIQ is broader, and represents what we describe as asynchronous operation. HTTP at the moment is a synchronous protocol. In other words, once a request has been made, the connection is maintained till such a time as the response returns. In the mobile scenario, disconnections are frequent, and so the existing protocol does not serve us well. We instead use WebIQ to make the process asynchronous. The user connects to the proxy, makes the request, and disconnects. When the user connects next, the result of his prior request(s) is available. S/he can then fetch this, and disconnect again. The user can connect again to provide the system feedback about the usefulness of this information. Thus the system manages disconnection. WebIQ allows the user to specify requests as queries, rather than explicit URLs. To obtain the results of these queries, the proxy first looks at its own cache, then queries its peer proxies, and only as a last recourse uses brute

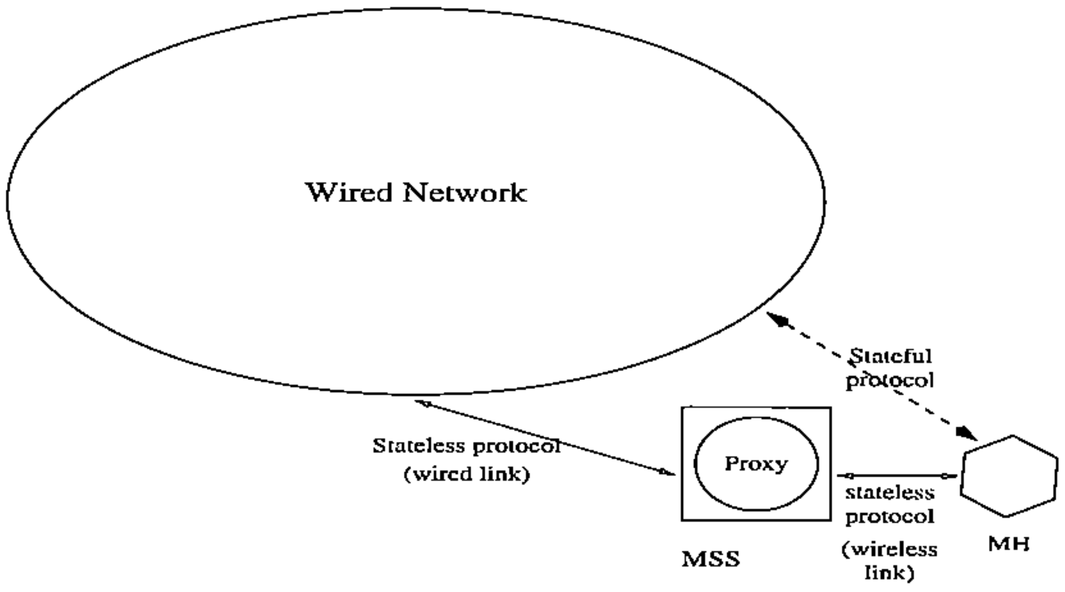


Figure 1: An abstract view of the architecture

force search engines (Yahoo etc.) available on the Internet. So our state information consists of data about the QOS from the network (bandwidth, latency), capabilities of the hardware in the MH, and the preferences and information requests of the user.

Proxy servers also have peer to peer protocols to exchange state information as MHs travel from the cell of one MSS to another. Essentially, our method employs an active join, passive leave technique[3] to initiate transfer of information. This means that when an MH decides (based on, say, signal beacons) that it should switch its MSS, it sends a message to the new MSS signaling a join with its own id and the id of its previous MSS. The new MSS can then request the profile of this MH/User, in terms of the information required by Mowser and WebIQ, from the old MSS. The other advantage of such peer protocols is that information about the state of the world obtained by one peer can be shared by the others. In fact, we view these multiple proxies as a distributed multiagent system which cooperates to solve certain problems. This aspect is explained in greater details in the section dealing with WebIQ.

The proposed software architecture allows us to address research issues such as

- The information needed by a user is spread across heterogeneous and geographically distributed systems. To expect the user to provide the location where the information resides is clearly not correct.

- The user may pose an extremely structured query (Jane Doe, "Some new developments in underwater basketweaving", J. UWBW Vol5, #1, pp 201-255), or an extremely unstructured one (Wasn't there an article on basketweaving by Jane Doe in the late 80's?).
- The response to a query from the user could involve aggregating information obtained from a variety of sources.
- The current models of information retrieval assume that the browsing software is operational after a request for information has been made. This is clearly not justifiable in the mobile scenario, where disconnections are a fact of life.
- While static networks are capable of sending and receiving large amounts of data without significant delay, this is clearly not the case in the mobile scenario. In addition to network bandwidth limitations and variability, mobile hosts also have relatively less resources than their static counterparts. These include smaller screen resolution and color capability, and limited or no sound capability.

Most existing approaches to disconnected browsing do not deal with these problems. They limit themselves to essentially caching information locally so as to be able to continue operation even if the network connection goes down. Our approach addresses these issues by augmenting information at each site with ancillary structures storing metadata about the state of the network, the capabilities of the MH and the preferences of the user. The cooperative retrieval process[19] will use a combination of knowledge–rich and knowledge–poor techniques. The system "dynamically" creates Web pages based on the client requests. In this sense, our architecture can be viewed as a system to develop multilevel models of hypertext [18] that are appropriate for disconnected browsing. We use the next two sections to describe in detail the Mowser & WebIQ systems, as well as their preliminary implementations.

4 Mowser

Mowser (Mobile Browser) allows a mobile user to set his or her viewing preferences, based on the network connection and available resources. Each mobile host's preferences are known to the proxy server. They are stored according to the mobile host's IP address, allowing the MH to access them from anywhere. The only requirement placed on the browser is that it be able to support proxy servers.

4.1 User Preferences

The preferences file contains information about the hardware capabilities of the MH, and its user preferences. The following information is stored for each MH:

- Starting Point - a default URL the MH will visit after preferences are saved
- Color Display- The color capability of the MH. Possible values are 8-bit or 24-bit, monochrome or color
- Video Resolution- Possible values are 640x480, 800x600, 1024x768
- Sound Capability- This will determine whether the MH has stereo, monotone, or no sound capability
- Text File Size- Maximum size of text files that may be downloaded
- Image File Size-Maximum allowed size for image files
- Reduce Image by- Preferred reduction technique for images (by image size/color)
- Video File Size- Maximum allowed size for video files
- Audio File Size- Maximum allowed size for audio files
- Unknown Type File Size- Maximum allowed size for files of unknown type

All of the settings have default values. A typical preferences file might contain the following:

```
Homepage= http://www.cs.purdue.edu;

Color= 8-bit;

Video Resolution = 640x480;

Sound Capability = Stereo;

TextSize (Kb) = 4;

ImageSize (Kb) = 8;

ReduceImageBy = size

VideoSize (Kb) = 8;

AudioSize (Kb) = 6;

UnknownSize (Kb) = 4;
```

4.2 Servers

Logically, the functionality of Mowser can be split along the lines of a preferences server and a caching server. The preferences server has two functions, to get the preferences for a MH at the start of a browsing session, and to update them whenever the user requests so. A browsing session is started on a MH by the Web browser contacting a preference server and saving preferences. These preferences may also be updated anytime during the browsing session. Such changes could reflect a change in network connection or availability of different resources during different sessions. Once the preferences are stored, the proxy server starts the browsing session by loading the starting URL defined in the preferences. All HTTP *GET* & *POST* requests from that point are handled by the proxy server. If a file requested by the MH does not meet the requirements described in the preferences, the proxy server can modify the file before sending it to the MH. The proxy server identifies the file type by means of its MIME type header. Depending on the type of the file, different conversions can be performed on it to meet the MH preferences. If a file does not meet the size requirement for its type, the proxy server will not send it to the MH. Files that cannot be properly displayed by the MH will not be sent either. A MH with no sound capabilities will not receive any sound files. If the MH has only monotone sound capability, stereo sound files would be converted to monotone, before they are sent. Image files can be compressed by sacrificing quality without sacrificing semantics. In particular, the size of the image, and the number of colors can be reduced, resulting in a smaller image that conveys the same information. Whenever a file is modified by the proxy server, a URL will be provided to retrieve the original unmodified file, if the user so requires.

4.3 Implementation

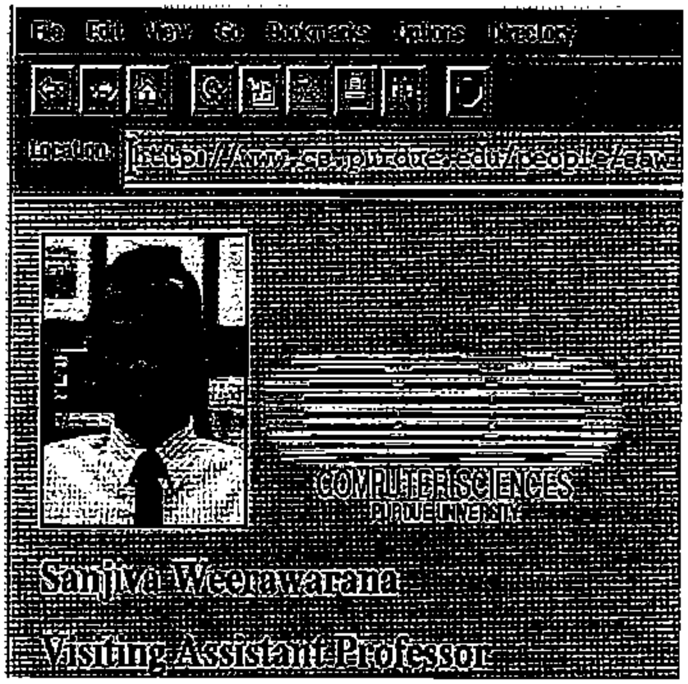
Our current implementation has two servers, running on different ports of a static host. One act as a preference server and the other as a proxy server. The preference server is implemented with NCSA HTTPD. The preferences are saved and updated using the CGI directory and are implemented as scripts written in Tcl and Perl. The proxy server is a modified version of an HTTP server written in Perl by Dr. George Vanecek[23]. This server was chosen for its simplicity and ease of modification. The proxy server performs *GET* and *POST* requests on behalf of the MH, and performs conversions on the received files if necessary, before sending them to the MH. Our current implementation performs conversions on image files if they do not meet the user's preferences. Both GIF and JPEG format files are subject to this conversion. The preferences allow the user to choose between reduction by image size or by number of colors. If the user decides colors are more important than the image size, no color loss will occur during the conversion.The file size is compared to the maximum size permitted in the preferences, and a scale factor is determined. The image file is then scaled down till it meets the size requirement. If however, the user has chosen to retain the image size during conversion, the image loses colors during conversion. If an imagemap is detected, then the user preference is overridden and the colors are reduced. This is because the coordinates associated with the callback from the imagemap are absolute, and scaling will destroy the link between locations on the image and the hyperlink to be followed. The converted file is then sent to the MH. Next to the modified image, a URL that points to the original image is added. If the image file is too large even after reducing to a predetermined minimum number of colors, an error message is sent to the MH in place of the image.

In ongoing work, we are linking in public domain media conversion formats for other multimedia types such as video. Also, we are rewriting the software as a single server written in Java which will provide for cross platform portability. Figure 2 illustrates the conversion done by Mowser on the homepage of one of the authors.

5 WebIQ

We view Web access as a sequence of small, atomic transactions. The connection between the MH and the proxy system need only be maintained during such transactions; there is no need for a continuous connection. These transactions involve retrieving information and providing metadata about the information. The proxy minimizes the bandwidth it uses over the mobile link as well as the power it forces the MH to consume. This is done by weeding out information which has a low probability of interest using user profiles. User profiles are thus important in determining what information will be useful. Users explicitly rate the URLs provided by the system, based on how useful they were. This gives the user control over the kind of information he or she would like retrieved, and allows the system to infer the user's profile. From another perspective, the user feedback allows the system to infer a *semantic* match between the user's query and the information in the document. Oft–repeated actions of users, e.g. fetching the stock market URLs every morning before 9 am, should be detected and automated. This would enable the MH to receive data during periods of low activity, saving time and CPU cycles.

WebIQ[14] uses existing information resources in a cooperative manner to find information. Each proxy will maintain a ranking of its peers, as well as external information brokers, based on the volume of information requested, quality value required for the results, and quality of previous responses etc [11]. Such information can be used to guide a query to a particular search engine, based on past information, or to a proxy which might be caching relevant URLs in response to a query by another user. Reusing previously obtained (and cached) URLs which were ranked highly by the user prevents a brute force keyword based search for every query. It also allows our metadata to indirectly capture the semantic content of the URL. In general, the proxy will contain the notion of user groups and use these to direct queries. Specifically, let us say that a user asks for URLs related to *football.* When rating the returned URLs, the rankings will be different depending on whether the user was European/Asian or American, since this word means different things to these


File Edit View Go Bookmarks Options Directory
Location: http://www.cs.purdue.edu/people/saw
COMPUTER SCIENCES
PURDUE UNIVERSITY
Sanjiva Weerawarana
Visiting Assistant Professor

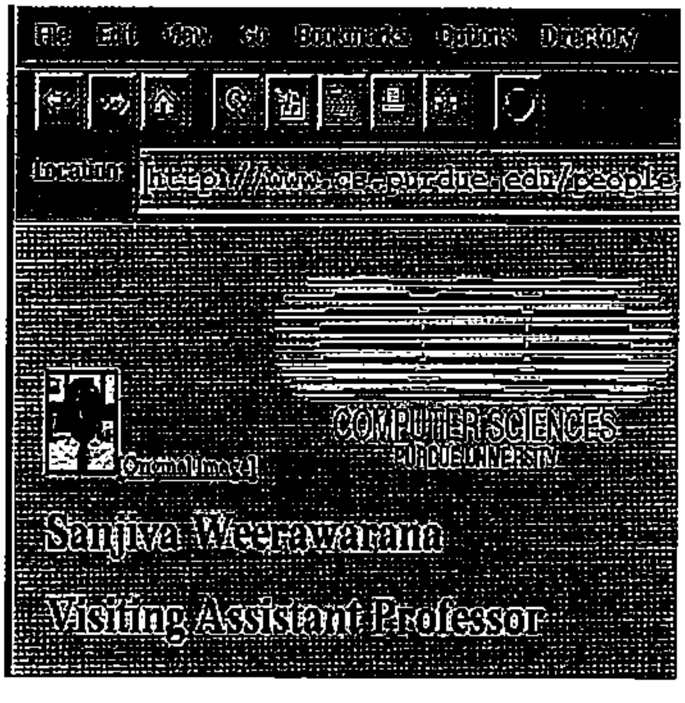

File Edit View Go Bookmarks Options Directory
Location: http://www.cs.purdue.edu/people
[Original Image]
COMPUTER SCIENCES
PURDUE UNIVERSITY
Sanjiva Weerawarana
Visiting Assistant Professor

Figure 2: Original and Converted pages using Mowser

two groups. Which rankings should a system use when another user queries for information related to football. Obviously, knowing what group the new user belongs to can help the proxy return appropriate cached URLs. This notion of groups and sharing information brings up the question of security and privacy. While we are interested in these issues, they are beyond the scope of the work we are proposing here. Figure 3 illustrates a sample of WebIQ's interaction with the user. The specific function of each part of the WebIQ system is discussed below.

5.1 User Functionality

The user connects to a web page containing an HTML form, which he uses to log on to and query the system. Once the query is registered, the user can disconnect and do other local work. At some later time, the user uses the same HTML form to log on to the system again. Once logged on, he receives a list of all pending results from previous queries. The user has the option to look at one or any number of them. Each result has a keyword, and a number of URLs associated with it. The user is asked to rate each URL (that he has seen) from the query on a scale, based on the degree to which it is appropriate for the particular metadata keyword(s) he queried on. A result is held until the user rates and submits at least one URL from all those received. All URLs which have not been rated at the time of submission of ratings are deleted.

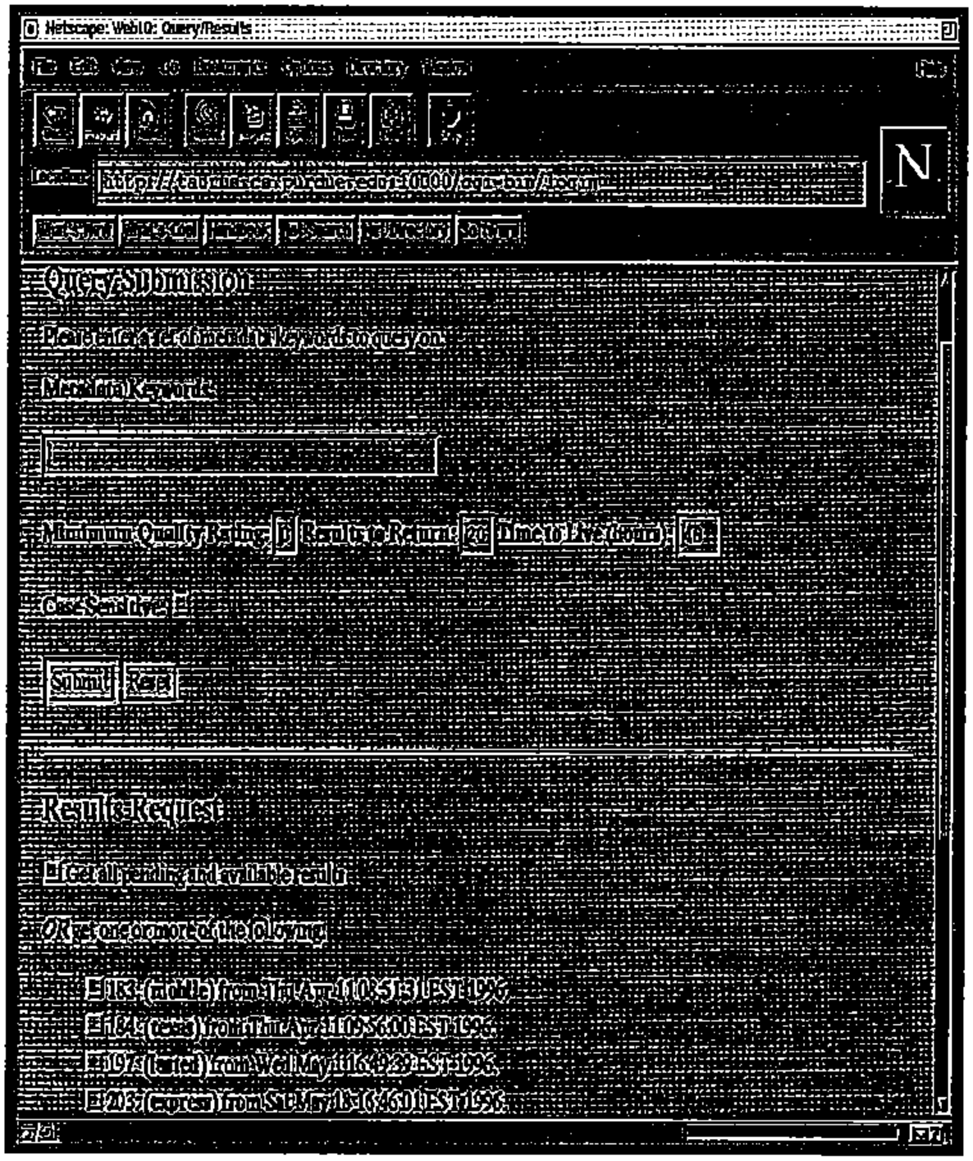
5.2 Filter Functionality

The filter's primary purpose is to communicate between the user interface and the server. It does this by interpreting the user requests received through the HTTP server, translating them into the query language, and sending them to the server. It also receives information in the query language from the server and translates it into a form suitable for viewing by the user (HTML).

5.3 Server Functionality

The server receives a request from the filter, and tries to answer the request. The request can be:

- to create an account,
- to log a user on to the server,



Location: http://taurus.cs.purdue.edu:10000/cgi-bin/[illegible]

Query Submission

Please enter a set of metadata keywords to query on.

Metadata Keywords:

Minimum Quality Rating: [1] Results to Return: [20] Time to Live (hours): [48]

Case Sensitive: ☐

Results Request

☐ Get all pending and available results

OR select one or more of the following:

☐ 183 (mobile) from Thu Apr 11 08:51:31 EST 1996
☐ 184 (texas) from Thu Apr 11 09:56:00 EST 1996
☐ 197 (tested) from Wed May 1 16:49:29 EST 1996
☐ 203 (express) from Sat May 18 16:46:01 EST 1996

Figure 3: WebIQ session

- to perform a query,
- to return results of a query,
- to rate links,
- or to change defaults settings.

These requests are answered in the WebIQ query language. In order to answer these requests, the server must maintain two different databases: a links database and a user profile database. Since it may also need to find information from outside sources, the server has interfaces to these information resources. Specifically, then, there are three main functions of the server:

- Storing and Serving URLs Using a Links Database The links database contains mappings from keywords to links. Each user has a memory space allocated for his query and link information. The links database is currently a public database, which can be searched by the WebIQ server, on behalf of other users. Each metadata keyword has a number of links to URLs that are associated with it. Each link has an information field associated with it containing a brief abstract about the link. Each URL <-> metadata keyword link has a *quality* weight associated with it. The choice of how to generate an initial rating is an interesting question by itself, though we are of the opinion that there can be no single method which will work for all users. The approach we followed was to determine the rating from a set of predefined data points. These data points are primarily generated from user ratings, although they can be generated in any number of ways, e.g. from the documents themselves, or from other information resources. When a query is received, the server looks for the documents with the highest weighted links to the metadata keywords given and returns these documents. URLs are primarily added to the database by getting results of user queries from either another WebIQ server or from an outside information source. They can, however, be entered into the database in other ways, including through user interest profiles (hotlists).
- Maintaining a Keyword <-> User Database We need a way of aggregating similar profiles of different users. This will enable a search based on profiles to be conducted. In our system, the

user profile revolves around the metadata keywords which each user uses for a query. In order to link together similar user profiles, we form a mapping from keywords to user names. When a user requests a search on a keyword, we append the user name to the end of a list associated with that key. Below, we show an example of how such an association could appear:

```
KEYWORD: mobile USERS: joshi@cs.purdue.edu, mike, todd@cs.purdue.edu
KEYWORD: basketball USERS: rrk@cs.purdue.edu
```

Note that a user can appear many times in this database, but only once per keyword. Also, each keyword can appear only once in this database.

- Interfacing with Other Information Resources : In order to answer queries that extend beyond its knowledge base, the server queries other sources of information. These sources are currently other web search engines, especially other WebIQ servers, but are not limited to search engines. The algorithm used to implement a lookup for links is given below: 1. The user's links database is queried for the required number of links 2. If step 1 fails, a search is made on the keyword<->user database. A list of other users is extracted, and a search is conducted in their links database. If the number of requested links is found, the results are returned to the user. 3. If the user's query cannot be satisfied using the databases of the local WebIQ server, the server tries to query other WebIQ servers that are known to it. 4. If step 3 fails to yield the required number of links, then the original WebIQ server tries to query other web information sources. Currently these are Infoseek and Yahoo, but any search engine can be used, given that a proper interface to it is written. All URLs that are received by the server from other information sources are entered into the links database, along with an initial quality rating. In this way, a WebIQ server for a group of users will, after a period of time, reflect the query interests of this group of users. Thus each server will acquire areas of expertise for which it knows where information can be found. After an initial "learning" phase, a WebIQ server can also act as an information source to other search engines.

5.4 Implementation

The implementation was split up into four modules. These were the filter-user interface, the server internals (parsing requests from the filter, database implementation etc.) , interfacing with other information resources, and interfacing with other WebIQ servers.

Filter - User Interface

The filter-user interface takes a form-based input from the user. This is translated into the WebIQ query language. The filter-user interface was intended to support the following interactive operation modes:

```
- Account creation.
- Login process.
- Query submission
- Result retrieval
- Rating process
- Default updating
```

The filter - user interface was implemented through "create" and "login" HTML documents that act as entry points to the system, as well as a number of scripts written in C++, one for each type of request that user can give. The WebIQ query language was designed to facilitate very short, atomic transactions between the filter and the server. Its structure is broken up into four subdivisions: UserName - this is included with every message packet, and is used to identify which user generated the query (or for which user the results are being sent). Password - carries the password of the user, necessary for user authentication. Information domain - this describes the type of information queried for. We split the different types of queries into four classes: administrative, query_data, links_data and automation_data. Information type - this describes the particular service that is being performed. The query types are based on the purpose of each query - e.g creation of accounts and login are administrative functions, involving access policy and usage restrictions. Query submission and results-requests are search oriented requests, and thus fall into a separate domain. Result generation was separated from results-request, and placed into a separate domain. This was to facilitate formatting of results in a hypertext based HTML document, so the user could directly walk off into links which were retrieved. Finally, automation data involves information about processes which the user initiates often enough that they can be automated, thus moving the burden of initiation from the user to the WebIQ system. We send the password either way as a consistent packet representation. We will discuss how this can be used for authentication later. In the following sample packets, all entries in bold are included for the user's understanding only, and are not contained in a message packet. The packet contains only those entries that are not in bold font. The WebIQ query format for the login operation is shown below:

UserName: xyz

Password: ***

Information domain: administrative

Information type: login

Expected reply for login:

UserName: xyz

Password: ***

Information domain: administrative

Information type: request_accepted/request_denied/error

Additional fields: If the login attempt was denied, this field contains the reason why it was denied. Otherwise, account information is sent back, a sample of which is:

Number of requests for which results are available: 1

Request identifier: 131

Number of keywords specified by the user: 1

Key to search under: java

Time when query was registered: Fri Dec 15 12:51:55 EST 1995

Minimum quality rating to retrieve: 0

Number of links to retrieve: 20

Time for which the results should be kept: 48

In practice, we send only the entries which are in normal Times font.

5.5 Server Internals

The server internals are comprised of a number of TCL scripts that manipulate the databases. The databases themselves are simple text files in the present instance. We are experimenting with porting these to a commercial database system.

Metadata Database

The Metadata database is organized as follows:

- Each user's profile information is stored in a separate directory for that user
- Each user's directory contains information about outstanding requests.
- The Metadata information of each user is stored in a file called MetaData.file, and the file holds data as records of the following type:

```
_WEBIQ_REQ_KEY boilermakers
_WEBIQ_REQ_OTHERKEYS
purdue
sports
_WEBIQ_REQ_LINKS
http://www.yahoo.com/Recreation/Sports/Basketball/College/Men/ 10
Recreation:Sports:Basketball:College:Men
_WEBIQ_REQ_INFO
last_count 0
new_count 0
update_period 48
time_to_live 48
start_date Thu Dec 14 11:28:18 EST 1995
```

The entry _WEBIQ_REQ_KEY holds the major keyword under which the search was conducted. "boilermakers" is the key which was searched for. There can be only one major key per Metadata record. The entry _WEBIQ_REQ_OTHERKEYS holds a list of related keywords, specified by the user at the time of the search. Each related keyword is separated from the rest by a newline. _WEBIQ_REQ_LINKS holds the links which were accepted by the user (i.e, those links which have been rated by the user). The links are stored in the following sequence: link ranking-for-link<newline> info-about-link<newline> link ranking-for... Thus links are stored first, followed by the user's ranking on the same line (but separated by whitespace), followed by information (a title) for the link on the next line. Finally, we have the entry _WEB_IQ_REQ_INFO, which holds maintenance data for each link, e.g. the time-to-live for the record, and when the keyword record was created.

File Locking & Mutual Exclusion: Since multiple users can be accessing the WebIQ system at the same time, we need to enforce mutual exclusion when reading and writing files. This was done by enforcing read/write locks around critical write sections. Locks were created using Unix file creation primitives. Whenever there were critical write sections, these locks were used as semaphores to provide mutual exclusion in the critical section.

The mapping from keywords to UserNames is stored in a directory called WebIqServerInfo. This directory holds management information for all users, including a user-password mapping file. The file named KeyToUser.map holds relations from keys to users. The results which have been retrieved need to be checked for repeated links. Results from within the WebIQ databases are pruned for repetitions, and results from other servers are pruned too. In the current implementation however, there is no effort to prune repetitions across local (i.e from the local WebIQ server) and global (from other servers) servers simultaneously. So there is still a possibility that links will be repeated. The retrieved results are parsed by the filter, and converted into a HTML hypertext document, which the user can then rate.

Query Status Information: The WebIQ server maintain status information as follows:

- Whenever a new request comes in, a file called keyWordFile<requestnumber> is created in the user's directory. <requestnumber> reflects the contents of a global counter for requests, at the time the request was logged. Obviously, the request counter forms a critical section which needs to be accessed using semaphore primitives. The keyWordFil<requestnumber> contains all the keywords the user has supplied for the query. During the search process, this list of keys can expand, depending on the values stored in the field _WEBIQ_REQ_OTHERKEYS for the keyword being searched. The first key in the list is always chosen as the major key to search under.

- infoFile<requestnumer> - This file is created in the user's directory. It contains information like "case sensitive(cs)" or" not case sensitive(notcs)", keywords need to be anded together, time search was started etc.
- the loginFIle contains login information about pending results.
- the tempLinkFile<requestnumber> - this contains all the unrated links generated by the search process. Once the links are rated, the links the user wishes to save will be stored in the file linkFile<requestnumber>.
- The Done<requestnumber> file is used to indicate that the search has been completed.
- The NumberOfRequestsLeft.file holds the number of pending requests for which searches are in progress/results are available.

5.6 Interfacing With Other Information Resources

The interfacing with other information resources is accomplished using perl scripts for each information resource, and a file-based interface with the server. For each keyword the perl scripts opens a socket and connects with an search engine server. All the search engines will be contacted with a keyword before a new keyword is extracted from the keyword file or until the number of links desired is reached. Currently the script will connect to either Infoseek or Yahoo. The results are saved in a temporary file in preparation to being parsed in order to extract the desired links. A difficulty arises in parsing the due to a lack of standardization. There is no standard to indicate that a link is a result of a query, or a link to "advertisements" that commonly show up in commercial search engines. Consequently, the parsing script must be modified to accommodate a new search engine. As each new URL is extracted, the results file is tested for an identical URL. If none is found, the URL is appended to the end of the file in the following format: <url> <title> <default rating> <keyword>. Should any problems occur such as the server disconnecting, the script will proceed to the next search engine.

6 Conclusion and Ongoing Work

In this paper, we present techniques that facilitate disconnected browsing – browsing the web from wirelessly connected, resource limited hosts typical of mobile systems. A software architecture suited for this task in particular, and for mobile information access in general, is described. Two components of this architecture are presented in detail, and their preliminary implementations described. One, called Mowser, handles the automatic transformation for multimedia data from the server to a form that the client can handle. The other, WebIQ, makes Web access an asynchronous operation which is relatively oblivious to disconnections. It also reduces the amount of useless (from the user's point of view) information that needs to be transmitted on the wireless link.

Our ongoing work has three major thrusts. The first relates to software improvement. We are trying to rewrite our software in Java to make it platform independent. Presumably, with the availability of Java compilers for many machines, performance will not be an issue. Along the same lines, we are trying to convert the file system based database used by WebIQ to use a commercial system such as Oracle. We are also enhancing the various media types that Mowser can handle and convert to bandwidth saving formats. The second thrust is to implement the peer–peer protocols for these systems. While we have defined these, the implementation is still not complete. Finally, we are currently deploying these systems internally and testing them with users. We would like to make them available more widely and conduct detailed studies of their utility in terms of ease of use, as well as the tradeoffs involved in our design choices.

References

[1] Grab a Site, *Grab-a-site product description*, http://www.bluesquirrel.com/store/116p.htm, 1995.

[2] A. Acharya, B.R. Badrinath, T. Imielinski, and J. Navas, *A www-based location-dependent information service for mobile clients*, http://www.caip.rutgers.edu/ navas/loc_dep_mosaic/Overview.html, 1995.

[3] B.R. Badrinath, A. Acharya, and T. Imielinski, *Impact of Mobility on Distributed Computations*, Op. Sys. Rev. **27** (1993), 15–20.

[4] T. Berners-Lee, R. Cailliau, J.-F. Groff, and B. Pollermann, *World Wide Web: The Information Universe*, Electronic Networking: Research, Applications, and Policy **2** (1992), no. 1, 52–58.

[5] W3 Consortium, *Http-ng problem statement*, http://www.w3.org/, 1995.

[6] P.M.E. DeBra and R.J.D. Post, *Information retrieval in the world-wide web: Making client-based searching feasible*, http://www.win.tue.nl/win/cs/is/reinpost/www94/www94.html, 1994.

[7] T. Drashansky, S. Weerawarana, A. Joshi, R. WeeraSinghe, and E. Houstis, *Software architecture of ubiquitous scientific computing environments*, ACM - BALTZER MOBILE NETWORKS AND NOMADIC APPLICATIONS (1996), (accepted for publication).

[8] D. Dreilinger, *Savvysearch*, http://www.cs.colostate.edu/ dreiling/smartform.html, 1995.

[9] G.H. Forman and J. Zahorjan, *The Challenges of Mobile Computing*, IEEE Computer **27** (1994), 38–47.

[10] Glomop, *Glomop description*, http://HTTP.CS.Berkeley.EDU/ fox/glomop/, 1995.

[11] A. Joshi, *To learn or not to Learn ....*, Lecture Notes in AI: Proc. IJCAI '95 Workshop on Learning and Adaptation in Multiagent Systems, Springer–Verlag, 1995.

[12] A. Joshi, R. Weerasinghe, S. McDermott, B. Tan, G. Bernhardt, and S.. Weerawarana, *Mowser: Mobile platforms and web browsers*, Bulletin of the IEEE Technical Committee on Operating Systems and Application Environments, 1996, Vol 8, no. 1.

[13] F. Kaashoek, T. Pinckney, and J. Tauber, *Dynamic documents: Mobile wireless access to the www*, Workshop on Mobile Computing Systems and Applications, December 1994.

[14] R. Kavasseri, T. Keating, M. Wittman, A. Joshi, and S. Weerawarana, *Web intelligent query - disconnected web browsing using cooperative techniques*, Proceedings of the First Intl. Conf. on Cooperative Information Systems, IEEE Press, 1996, pp. 167–174.

[15] Y. Lashkari, *Webhound*, http://webhound.www.media.mit.edu/projects/webhound/, 1995.

[16] NEXOR Ltd, *Introduction to aliweb*, Nottingham, UK, 1995.

[17] Milktruck, *Milktruck delivery description*, http://www.milktruck.com/cpsisv/index.htm, 1995.

[18] J. Myfield and C. Nicholas, *Snitch: Augmenting hypertextdocuments with a semantic net*, Intl. Journal of Intelligent and Cooperative Information Systems **2** (1993), 335–351.

[19] T. Oates, M.V. Nagendraprasad, and V. Lesser, *Cooperative Information Gathering: A Distributed Problem Solving Approach*, Tech. Report TR-94-66, UMASS, 1994.

[20] J.E. Pitkow and Mimi Recker, *Integrating bottom-up and top-down analysis for intelligent hypertext*, 1994, GVU Center, College of COmputing, Georgia Tech.

[21] B.N. Schilit, N. Adams, R. Gold, M. Tso, and R. Want, *The parctab mobile computing system*, Proceedings of the Fourth Workshop on Workstation Operating Systems, October 1993, pp. 34–39.

[22] Harvest System, *Harvest system*, http://harvest.cs.colorado.edu/, 1995.

[23] George Vanecek, *Perl web server code*, Personal Communication.

[24] G. Voelker and B. Bershad, *Mobisaic: An information system for a mobile wireless computing environment*, Workshop on Mobile Computing Systems and Applications, December 1994.

[25] WebWatch, *Webwatch product description*, http://www.specter.com/, 1996.

[26] WebWhacker, *Webwhacker product description*, http://www.ffg.com/whacker.html, 1995.