1996

Web Intelligent Query - Disconnected Web Browsing Using Cooperative Techniques

Ramanathan Kavasseri

Todd Keating

Michael Wittman

Anupam Joshi

Sanjiva Weerawarana

Report Number:
96-002

WEB INTELLIGENT QUERY - DISCONNECTED WEB BROWSING USING COOPERATIVE TECHNIQUES

Ramanathan Kavasseri
Todd Keating
Mike Wittman
Anupam Joshi
Sanjiva Weerawarana

Purdue University
Department of Computer Sciences
West Lafayette, IN 479807

CSD TR-96-002
January 1996
(Revised April 1996)

Web Intelligent Query - Disconnected Web Browsing Using Cooperative Techniques

Ramanathan Kavasseri, Todd Keating, Michael Wittman, Anupam Joshi, Sanjiva Weerawarana
Department Of Computer Sciences
Purdue University
{rrk,keatinwt,wittmamr,joshi,saw}@cs.purdue.edu

Abstract

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. If at the time it "goes down", a mobile computer is involved in a transaction process with another computer (mobile or static), it should be able to tolerate the "fault" of temporary disconnection. This work focuses on disconnected 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 presents a preliminary implementation.

1.0 Introduction

Recent years have seen a significant increase in interest generated by mobile computing. Users are increasingly looking for mobile aware applications[2][3]. With the advent of the World Wide Web[13], browsing (surfing) the web is becoming an increasingly common activity for computer users.

The current model of browsing the web leaves most of the burden of finding the relevant information on the user and his computer. The user is expected to know starting points, and then search through documents to find the information s/he needs. While the user searches for information, a network connection needs to be constantly maintained. Moreover, a lot of (potentially) useless information is transmitted over the 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. Wireless networks typically provide far lower bandwidth than 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 [14] for details of the mobile computing scenario. Providing a reasonable level of performance in the face of frequent disconnections and restricted bandwidth is a major issue in mobile computing. In this paper, we describe a system called WebIQ which facilitates disconnected mobile browsing.

A widely accepted model of operation in the mobile computing domain is the MSS-MH model[12]. In this class of models, each *Mobile Host* (MH) is assumed to have a wireless link to a *Mobile Support Station* (MSS). The MSS acts as a gateway to the wired world. Based on such a model, we investigate in this work the effects of an asynchronous mode communication protocol, implemented by a proxy server system we call WebIQ (Web Intelligent Query), in offsetting the effects of the mobile environment. WebIQ runs on MSSs, and provides an interface to the Web for a browser on a MH. The objective of the WebIQ system is to allow a different model of web access suited for disconnected browsing. In this model, we view web access as a sequence of small, atomic transactions. The connection between the MH and the WebIQ system need only be maintained during such transactions, there is no need for a continuous connection. These transactions consist of the following functions:

- Providing WebIQ with a request for information and parameterize information quality
- Retrieving appropriate links provided by WebIQ
- Extracting information from links as desired
- Providing WebIQ ratings about the quality of the info-response to a query (responses are quality-rated URLs).

Such a model, in conjunction with local caching of information and media transformation [15], will prove useful for disconnected browsing. We posit that any system which supports disconnected browsing by mobile hosts should meet the following requirements:

- operation in the presence of frequent disconnections.
- minimizing wireless bandwidth usage and consequently MH power consumption - this can be done by weeding out information which has a low probability of use at the MSS itself using user profiles.
- usefulness/quality rating of retrieved URLs - this gives the user control over the information he would like retrieved, and allows the system to infer a user's preferences.
- detection/automation of oft-repeated user patterns, e.g. fetching the stock-market URLs every morning before 9am, thus enabling information retrieval during periods of low activity, saving time and CPU cycles.
- cooperative information gathering - this requires maintaining a ranking of WebIQ peers, as well as external information brokers, based on the volume of information requested, quality value required for the results, etc. Such information can be used to guide a query to a particular search engine, based on past information.
- amortizing the cost of information retrieval and reducing search-time by caching URLs at the MSS, thereby redirecting queries away from overloaded information resources.
- simple user-interface, requiring nothing more complex than forms-capable browers.

The WebIQ system was thus developed with the following goals in mind:

- to enable disconnected browsing without impeding browsing on static hosts.
- to create a network of WebIQ servers, with each WebIQ server handling queries from a small group of MHs.
- to create a user profile for each user, based on associations between keywords (which are used for searching) and the URLs retrieved for these keywords.
- associating a quality rating with each metadata keyword <-> URL association in each user's profile.
- to use a cooperative information gathering strategy that first queries information resources close to the WebIQ server, especially other WebIQ servers, then moves on to more central sources of information if necessary.
- to allow use from any forms capable browser.

2.0 Related work

There is very little extant work in the area of disconnected browsing. Most such work has been limited to the notion of caching URLs using some kind of fish search. 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 part view this as an information retrieval problem. Little previous effort has been directed at adapting the underlying model of web browsing, which is flawed from a mobile context, to disconnected browsing.

Oates et al. [8] have concentrated on cooperative information gathering. This information gathering (IG) technique involves the pro-active acquisition of information, but may require analysis of intermediate results and a satisficing search.

SavvySearch[16] allows parallel searching on multiple information resources. The user chooses information resources to be searched. This architecture does incorporates parallelism, but still requires brute-force searching for each query. No attempt is made to categorize the information in order to develop an efficient method of searching.

NEXOR's ALIWEB[6] requires a site to create a file, containing a description of their services in a standard format. The site then registers with ALIWEB, which routinely retrieves them and creates a searchable database of the services. This is a method of categorizing information, but it does not allow any details about the quality of the services. The Harvest architecture[18], like ALIWEB, requires sites to create a resource index. A gatherer then retrieves these indices and passes the information, after filtering, to brokers. A search is then conducted by con-

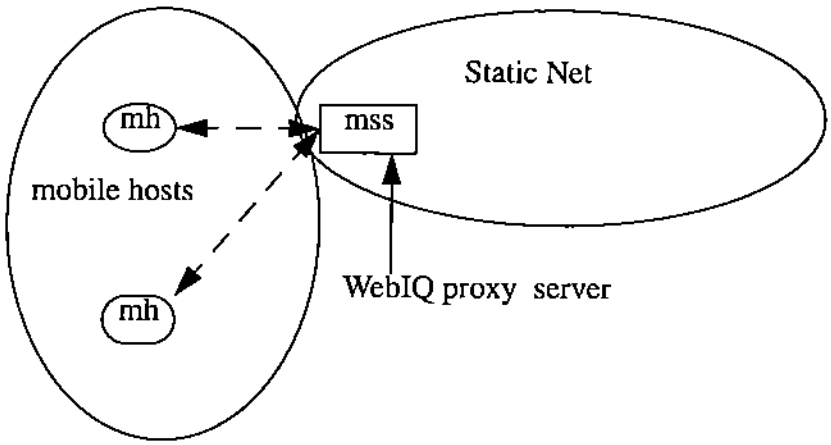

Figure 3.1

tacting a broker who may have the required information or can contact other brokers if necessary. Like ALIWEB, Harvest lacks relevance feedback.

WebHound[17] requests quality rankings on retrieved documents, thus implementing user profiling. WebHound, however, does not act as a web searcher based on these quality ratings.

DeBra and Post[4] suggest the use of the fish-search algorithm. The fish-search algorithm is an IG type of search wherein documents relevant to users' requests are recursively searched for links to other documents. Searching links from a relevant document uses less resources than brute force searches, and is more likely to provide relevant documents.

Pitkow and Recker[7]'s study of document access and usage patterns confirms the ability to identify user patterns for use in prefetching documents or automating temporal events.

A combination of the above methods can lead to a more effective web search engine, which, along with disconnected browsing, are the major goals of WebIQ, proposed in this paper. The WebIQ system focuses on bandwidth conservation by retrieving only a prespecified number of links over the MSS-MH link. Such links are selected based on a user profile, as in WebHound. User profiling facilitates temporal pattern identification, thus opening the door to possible information prefetching and automation of services. WebIQ implements a hierarchical system of collaborative information sharing to reduce congestion (and consequently, delays) at existing web-searchers like Yahoo. We also propose information sharing among WebIQ clients, in the hope of amortizing information retrieval costs.

3.0 WebIQ design specifications

3.1 WebIQ software architecture

The software architecture of the WebIQ system is based on the model [12] shown in Figure 3.1. The MHs communicate to the Static internet through MSSs. On the static net bandwidth (volume of information transmitted) is not too big a concern, as compared to the same over a wireless channel. Hence a MSS can receive large volumes of data from the Internet, but must transmit conservatively over the wireless link to a MH. A MSS filters out unnecessary information retrieved from the static net before passing the information on to a MH. Thus a MSS is a client to a server on the static net, but acts as a proxy server to the MHs associated with it.

The WebIQ project has been divided into three components - the user, the filter, and the server. The user can be anyone on the Internet (provided that they use a HTML forms-capable browser). The filter acts as the user interface. The filter interprets all user requests, which it receives from an HTTP server via CGI. The filter then translates the user request into an internal query representation [9,10] used by WebIQ servers. The server receives the request from the filter and acts on it. The server maintains its own database from which to answer queries, but also connects to other servers and information resources on the Internet when necessary.

3.2 WebIQ functionality

The specific function of each part of the WebIQ system is discussed below.

3.2.1 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 reloads (or reconnects to) the HTML form, whereupon he receives a list of all pending results from previous queries. Each result has a keyword, and a number of URLs associated with it. The user is asked to rate URLs of interest based on the degree to which it is appropriate for the particular metadata keyword(s) he queried on. Unrated URLs are discarded. Thus a step-by-step procedure a typical user would follow in the WebIQ system would be:

1. Log on to the system with a password/create a new account and then log on.
2. Specifying keyword(s) to search on. At this time the user must also specify the minimum quality level of links he wants retrieved (new links are rated as a default 5, until the user changes their rating).
3. Specifying the maximum number of URL links to return.
4. Submitting the query. (Note that till this point, the user follows exactly the same procedure that he would with any other search engine).
5. Now the user is free to disconnect from WebIQ and do other work. At a later time, the user will reconnect by clicking on "reload" or logging on again.
6. When the user reconnects, he sees a list of all previous queries, and he can click on any query to view the results (the results are formatted as a HTML document that the user can easily follow).
7. Next to each URL in the result listing, the user will also see a list of radio-buttons numbered from 1 to 10. These are for quality rating feedback, and the user can click on one of these buttons, if he decides to rate the links.
8. Finally, after viewing his results, the user needs to specify the time -to-live for the rated results of this query.

3.2.2 Filter functionality. The filter's primary purpose is communicating between the user interface and the server. It does this by translating the user requests (received by forms submission) into the query language, and sending them to the server. It also receives information from the server and translates it into a form suitable for viewing by the user (HTML).

3.2.3 Server functionality. The server recognizes requests from the filter:

- to create an account,
- to log a user on to the server,
- to perform a query,
- to return results of a query,
- to rate links,
- or to change defaults settings.

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 Retrieving URLs Using a Links Database

The links database contains mappings from keywords to links. The links database is a public database, which can be searched by the WebIQ server, on behalf of other users. This database is implemented as a set of tables, reduced to the third-normal form [19]. The first table contains a mapping from keywords to links, with a quality rating for each keyword-URL pair (note that the same URL can have a different quality rating when associated with another keyword). One such table is allocated to each user. When a query is received, the server looks for the document URLs with the highest weighted links to the metadata keywords given and returns these URLs. URLs are primarily added to the database by getting results of user queries from either another WebIQ server or from an outside information source.

We maintain a separate table associating each retrieved URL with information like time elapsed since last use, use-count, etc.

- 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. Note that a user name 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 other web search engines, especially other WebIQ servers. 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.

URLs retrieved by the server are rated and stored in 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.

4.0 Implementation

The implementation was split up into four modules. These were:

- the filter-user interface,
- server internals - parsing requests from the filter, database implementation etc.,
- interfacing with other information resources, and
- interfacing with other WebIQ servers.

4.1 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 is implemented through HTML documents and CGI scripts. 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 parts:

UserName - used to identify which user generated the query.

Password - necessary for user authentication.

Information domain - this describes the type of query, which is split 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 , and fall into four separate domains, e.g creation of accounts and login are administrative functions, involving access policy and usage restrictions. Query submission and results-requests are classified as query_data and links_data, respectively. 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. The query domain information is stored in the Information domain field of each packet. We send the password either way as a consistent packet representation. The Information type field distinguishes the subdomain that the query addresses, e.g. - the administrative domain has account creation, account login and account default information as subdomains. For further information on the internal query language, the reader is referred to [9] and [10].

4.2 Server internals

The server internals are comprised of a number of ORACLE scripts that manipulate the databases. The database is currently being ported from simple TCL managed files to Oracle [19]. We will explain the Oracle implementation in this work.

4.2.1 Links database. The Links database uses a table representation to map a Keyword-URL pair to its user-specific information. It is implemented using commercial software (to improve search times and allow for future upgrades). Each user has a table mapping keywords (from previous queries) to links. Each such entry has a rating associated with it, an access count to tally the number of times the user accessed this link, a time-to-live and a time-stamp for the time of last access. The fields of this table are thus : "KEYWORD", "URL", "RATING", "ACCESS-COUNT", TIME-TO-LIVE", "LAST-ACCESS", which is the simplest relation between a keyword-URL pair and its associated metadata information, especially quality rating. For example, a single entry could be **"keyword = boilermakers", "URL = http://www.yahoo.com/Recreation/Sports/Basketball/College/Men", "rating = 10", "access-count = 3", "Time-to-live = 2 Days","last-access = Thu Dec 14 11:28:18 EST 1995"** .

We maintain another table mapping links to information about them, like a string describing the associated URL, time-to-live (which is the maximum of the TTLs specified in each keyword-URL mapping the that link is associated with), and access count. This is useful in generating statistics about system usage.

4.2.2 Mutual exclusion & data consistency. Since multiple users can be accessing the WebIQ system at the same time, mutual exclusion must be enforced when reading and writing data. In our implementation, data consistency is maintained by the commercial database software, (Oracle).

4.2.3 The key to user map & other tables. The mapping from keywords to UserNames is stored in a separate table common to all users. A password table stores the mapping from users to their encrypted passwords. Further, system management tables hold information for all users, like usage statistics, information about other servers, etc.

4.2.4 Format of retrieved results. The results which have been retrieved need to be checked for repeated URLs, as they have (possibly) been compiled from multiple, independent sources. Results from within the WebIQ databases are pruned for repetitions, and likewise results from other servers. In the current implementation however, there is no effort to prune repetitions across local (i.e from the WebIQ servers) and global (from other servers) servers simultaneously. So there is still a possibility that links will be repeated (by being retrieved from the WebIQ system and also retrieved from central information resources like Yahoo). The retrieved results are parsed by the filter, and converted into a HTML hypertext document, which the user can then follow and rate.

4.2.5 Query status information. The WebIQ server maintains status information as follows:

1. Whenever a new request comes in, a unique identifier is created for it. All the keywords the user has supplied for the query are associated with this identifier. During the search process, this list of keys can expand if related keys are found (this is done by requesting keywords using URL links as the search key - a dual to the problem of searching links from keywords - as such, this is done only if search requests fail to retrieve appropriate links).

2. Associated with each request (i.e. the keywords to search) are the and/or boolean combinations (which are not yet fully supported by WebIQ), and reqested number of links. Also, work is in progress to permit the user to search specific information resources, as is allowed in Savvy Search.

3. User-specific information is kept about pending results from previous queries, and is sent to the user each time he logs on/reloads the forms document, so he knows the status of the queries.

4.3 Interfacing with other information resources

Interfacing with other information resources is accomplished using a customised interface for each information resource. For each keyword the script opens a socket and connects to a search engine. For each keyword, we cycle through all the search engines in our list, before cycling to the next keyword. We continue cycling until we either run out of keywords, or we retrieve the desired number of goals. Currently the server will use Infoseek or Yahoo, but the addition of other information resources is simple, as we maintain a consistent interface between the server and script. We also provide a degree of fault-tolerance by allowing connections to multiple resources, so should one fail, we can skip to another information resource.

4.4 Interfacing with other WebIQ servers

Interfacing with other WebIQ servers is fairly easy, since WebIQ speakers speak the same query language. We use HTTP to talk between servers, similar to the working of the filter.

5.0 Conclusion

The WebIQ system is a tool to support disconnected browsing. It provides a mechanism to implement the dis-

connected browsing model. At another level, however, it can also be viewed as a sophisticated cooperative search engine for the Web. Unlike most search engines, the user can specify in advance the number of links he wishes to retrieve. This, coupled with disconnected browsing, make the WebIQ system very appealing for mobile computing. Disconnected browsing means that the user no longer needs to remain connected to the search engine while a search is being conducted. The ability to specify a maximum number of links means that we wont be flooding the mobile host with tons of links which the user will never access - battery lifetimes for mobile hosts last typically for a couple of hours, and the amount of web surfing that can be done is limited in such a short time span. WebIQ focuses on retrieving links based on past user history. The user can also associate a measure of usefulness with each retrieved link, and can cache the links he likes for future reference. The user can specify minimum rating values (i.e a user can instruct the WebIQ system not to return links below a particular rating value). This ensures that the user will not receive a flood of unwanted results.

Thus we have introduced features to make the WebIQ system powerful for mobile environments, namely:

1. disconnected browsing
2. cooperative information sharing within users of a particular system
3. accessing remote search engines based on the volume of links requested.

The current implementation of WebIQ, however, still does not faithfully implement all its design objectives. In our continuing work, we hope to address the following constraints imposed by the current implementation:

1. There is a lack of boolean operators which can be used to link keywords, e.g a user cannot "and" two keywords, and combine them into a single search request. The only mode of operation supported is a hierarchical ORing, where the links for the first keyword are searched, then links for the second keyword,etc. To account for boolean operators, the keyword - URL relationship should be replaced by a "query - URL" relationship, which will require slightly more complex algorithms to match query patterns.
2. The user should be able to delete all the links retrieved, without rating them. At present, the user is forced to rate at least one link.
3. There should be an efficient way to browse through a user's links database.
4. Cooperative browsing among WebIQ servers - although the interface for this feature has been written, it has not been linked to the query process(i.e to the search algorithm) Right now, the search method searches cooperatively within a server, and the jumps out to non-WebIQ servers directly.
5. The user might want to specify that he wants to skip searches within his server, or that he wants to search only Yahoo, or Yahoo and Infoseek, etc. All these access methods must be provided to the user.
6. Privacy issues - We are considering how to incorporate privacy into the database. Users may not want certain links/keywords/data to be accessible to others, and a means of locking such information needs to be incorporated.
7. Scalability - to ensure scalability, we need to delete cached information periodically. The framework for this is in place (time-to-live, time-stamps and access counts). Using these, we plan to test various cache pruning strategies, like least-recently used, least frequently accessed, etc.

Acknowledgments

This work was supported in part by NSF award ASC 9404859, and a grant from the Intel Corporation. The authors wish to thank Peeyush Ranjan for valuable tips on improvements to the WebIQ user interface (and being the first user of the system), and Shalab Goel for information on database management.

6.0 References

1. Bowman, C. M. et. al. The Harvest Information Discovery and Access System. *Proceedings of the Second Intl. WWW Conference*, Chicago, 1994, 763-771
2. Drashansky, T., Weerawarana, S., Joshi, A., Weerasinghe, R., Houstis, E. Software Architecture of Ubiquitous Scientific Computing Environments for Mobile Platforms, Technical Report CSD-TR-95-065. Department of Computer Sciences, Purdue University.
3. Joshi, A., Weerawarana, S., Drashansky, T., Houstis, E., SciencePad: An Intelligent Electronic Notepad for Scientific Computing, *Proc. Intl. Conf. Intelligent Information Management Systems*, Washington, D.C., 1995, 107-110.
4. DeBra, P.M.E., and R.D.J. Post. Information Retrieval in the World-Wide Web: Making Client-based searching feasible. Eindhoven University of Technology Department of Computing Science. Netherlands.
5. Kent, Robert E., and Christian Neuss. Conceptual Analysis of Resource Meta-information.

6. NEXOR Ltd. Introduction to ALIWEB. Nottingham, UK. 1995

7. Pitkow, James E., Recker, and Mimi Recker. Integrating Bottom-Up and Top-Down Analysis For Intelligent Hypertext. Georgia Institute of Technology College of Computing Graphics, Visualization &Usability Center. Atlanta. 1994.

8. Tim Oates, MV Nagendra Prasad, and Victor R. Lesser. Cooperative Information Gathering: A Distributed Problem Solving Approach. UMass Department of Computer Science. Amherst. 1994.

9. Ramanathan Kavasseri, Todd Keating, and Mike Wittmann. WebIQ internal document for server-filter communication. http://www.cs.purdue.edu/wittmamr/WebIQ/filter-server.html.

10. Ramanathan Kavasseri, Todd Keating, and Mike Wittmann. WebIQ internal document for server-server communication. http://www.cs.purdue.edu/wittmamr/WebIQ/server-server.html.

11. A. Joshi, "To Learn or not to Learn ...", *IJCAI Workshop on Learning and Adaptation in Multiagent Systems*, Montreal, 1995 (to appear in Springer's LNAI series).

12. T. Imielinski and B. Badrinath, Mobile Wireless Computing: Challenges in Data Management, *Communications of ACM* **37** (1994), 18-28.

13. 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), 52-58.

14. G. Forman and J. Zahorjan, The Challenges of Mobile Computing, *IEEE Computer*, April 1994.

15. A. Joshi, R. Weerasinghe, S. Weerawarana, Mowser: A Web Browser for mobile platforms (manuscript under preparation), http://www.cs.purdue.edu/research/cse/mobile/mowser.html

16. D. Dreilinger, SavvySearch, http://www.cs.colostate.edu/~dreiling/smartform.html

17. Y. Lashkari, WEBHOUND, http://webhound.www.media.mit.edu/projects/webhound/

18. Harvest System. http://harvest.cs.colorado.edu/

19. George Koch & Kevin Loney, "ORACLE, The Complete Reference", 3rd Edition, Osborne-McGraw Hill.