December 2002

XSAR: XML BASED SEARCH AGENT FOR INFORMATION RETRIEVAL

Akhilesh Bajaj
Carnegie Mellon University

Hideaki Tanabe
Carnegie Mellon University

Chao-Chin Wang
Carnegie Mellon University

XSAR: XML BASED SEARCH AGENT FOR INFORMATION RETRIEVAL

Akhilesh Bajaj
The H. John Heinz III School of Public Policy and Management
Carnegie Mellon University
akhilesh@andrew.cmu.edu

Hideaki Tanabe
The Internetworking Institute
Carnegie Mellon University

Chao-Chin Wang
The Internetworking Institute
Carnegie Mellon University

Abstract

The current form of information on the world wide web (WWW) is mainly in the form of unstructured information on scattered web sites. The usual method of accessing this information is using keyword-based or directory-based search engines. Recently, an increasing number of organizations, bodies and associations are adopting XML (extensible markup language) document type definitions (DTD) with a view towards putting structured information on the WWW. As information becomes more structured, we anticipate the spread of large XML repositories that will be accessible on the WWW. To help search these repositories, we present a novel search tool: XSAR (XML Based Search Agent For Information Retrieval), to access these repositories. We describe the architecture and functionality of XSAR, analyze its performance along four metrics and compare it to alternate existing search mechanisms. This work demonstrates the feasibility of agent based search mechanisms on large information repositories.

Introduction

Over the last decade, the number of hosts and the quantity of information being stored on the world wide web (WWW) have both been increasing significantly (see http://www.isc.org for updated statistics). This information is stored on web sites in the form of different file types, *e.g.,* various alphanumeric, video and audio formats. It is well known that much of the information is still stored in the form of Hypertext Markup Language (HTML) files. Recently, there has been a great deal of interest in storing information in the form of extensible markup language (XML) files [1]. XML allows the information to be **structured**, much like in a relational database. Several researchers believe that XML will allow the web to become more semantic in nature, where agents will be able to access information and understand what it means [2]. Various bodies are currently at work specifying XML document type definitions (DTD) for their domains of interest, *e.g.,* Dublin Core for library science (http://dublincore.org), Chemical Markup Language (CML) (http://www.xml-cml.org), Rosetta Net (http://www.rosettanet.org) for supply chain management in the electronics industry and News ML (NML) (http://www.newsml.org).

The storage mechanisms for WWW information are also varied, ranging from pages served by scattered WWW sites, to a collection of pages served by a single WWW site (such as a newsgroup or a portal site like http://www.yahoo.com[1]) to dynamic pages generated from information that is stored in a relational or object-relational database.

[1]Throughout this work, when we refer to a single web site, we accept that the site may be served by multiple web servers for scalability reasons.

Figure 1 shows the different types of information formats and possible methods of storage for these formats.

Information Format → / **Storage ↓**	Static HTML Pages	Dynamically Served Alphanumeric Data	Static XML Data
Scattered WWW Sites	X		X
Single WWW Site	X		X
Database		X	

Figure 1. Information Formats and Their Possible Methods Of Storage

As the amount of information has increased, various search mechanisms and tools have been created for accessing this information. These include: a) search engines like http://www.altavista.com and http://www.google.com, that employ either a centralized database or a centralized directory scheme for HTML pages across the web, b) peer-to-peer file systems like http://www.napster.com for HTML files and multimedia content and c) site-specific dynamic querying capabilities employing available technologies like CGI (Common Gateway Interface), ASP (active server pages) or JSP (java server pages).

Figure 2 shows the different kinds of search mechanisms that are available and the kinds of information formats that can be retrieved by them.

Web search engines that search files on scattered web sites have their ancestors in the information retrieval (IR) systems developed during the last fifty years. IR methods include Boolean search methods, vector space methods, probabilistic methods, and clustering methods [3]. All these methods aim at finding the relevant documents for a given query. For evaluating such systems, *recall* (the number of relevant retrieved documents divided by the total number of available relevant documents) and *precision* (the number of relevant retrieved documents divided by the number of total retrieved documents) are the most commonly used measures [4]. The *percentage of dead links* found is also a useful metric, and is usually dependent on how accurately the universe of pages[2] listed in the search engine's database or directory structure reflects the real existence of the same pages. Finally, *response time* (time taken for the user to get the information she desires) has been found to be a useful metric in several works that deal with human computer interaction [5].

Information Format → / **Search Mechanisms ↓**	Static HTML Pages	Dynamically Served Alphanumeric Data	Static XML Data
Search Engines using Key Words	X		
Directory Based Search Engines	X		X
Site Specific Query Mechanisms	X	X	X
Peer-To-Peer	X		X

Figure 2. Information Formats and Their Possible Methods of Search

Furthermore, the performance of each search tool is directly influenced by the user's ability to more narrowly define the nature of the query, *e.g.,* in the form of more precise key-word strings or correct selection of a higher level directory classification. For peer-to-peer systems, the search is usually based on file names, which are supposed to reflect the underlying content (*e.g.,* the name of a song or an artist represents the content). The same metrics apply here also. The primary drivers of performance in keyword and directory based search engines is the coverage and searchability of the underlying central database or directory

[2]In this work, we define the universe of pages to be the references to actual pages; where the references are stored in the underlying centralized database or directory structure of the search engine.

scheme that centrally stores information on the universe of documents; while in peer-to-peer networks performance is driven by algorithms that drive caching as well as the routing of search strings from peer to peer.

Several works (see [6] for a summary) suggest that the storing of information on widely scattered sites poses serious problems to the scalability, precision and recall of the relevant search methods and tools. This is because the underlying central database or directory scheme of the search engines used to search for this information are not scalable. Thus, as the number of hosts and pages increases, these search methods will take longer, yield poorer precision and recall and place greater burdens on users to specify queries narrowly. Also, an increasing number of dead links will appear in query results, since the central database or directory scheme is not synchronized with the existence of the actual pages.

Since there are so many scalability problems with accessing information that is stored on scattered web sites, we envision a growing trend towards the creation of information repositories, which we define as a collection of documents that have a similar structure, and are served off a single web site (with possibly multiple servers). Examples would include repositories of newspaper articles, repositories of articles dealing with, say, physical chemistry research, etc.

While peer to peer architectures offer a potential solution to accessing information on scattered web sites, they also have limitations in their current forms: a) the limits of content-richness in file-names and META tags (since searching is currently done using file names and META tags) and b) the tradeoff between network traffic and recall [7]. Thus, while peer-to-peer mechanisms are useful for searching for multimedia content (*e.g.,* songs by a particular artist), they are still not very useful for detailed content-based querying (*e.g.,* newspaper articles written by author *A* and about Subject *S,* written between dates *D1* and *D2*).

Given the increased acceptance of XML as a standard, the potential of XML to structure the information available on the WWW, and the increasing emergence of DTDs in several domains, we envision a shift in the methodology of information storage on the WWW. Specifically, we see an increasing amount of information being stored in repositories of XML files. For example, chemical structure information on drugs is already being stored in a CML repository at http://www.ch.ic.ac.uk/chimeral/resources/cml/drugs/index.html. It is fairly reasonable to imagine news organizations offering repositories of news articles using, say, the NML format in the near future, libraries offering books using, say, the Dublin Core standard and players in an electronic parts supply chain each offering their own repository of information following, say, the Rosetta Net standard.

The primary contribution of this work is the description of a dynamic, DTD-independent query agent that we have created called XSAR (**X**ML Based **S**earch **A**gent for Information **R**etrieval). Essentially, XSAR can be used to query information repositories of XML documents (see figure 1). The XSAR web site is at http://bajaj.heinz.cmu.edu/XSAR/Xsar/.

The rest of this paper is organized as follows. Section 2 describes the functionality and high level architecture of XSAR. Section 3 describes the low-level architecture of XSAR, and a description of how it was built. Section 4 lists performance issues related to XSAR, and compares it to other possible query mechanisms. Section 5 concludes with limitations and future work.

Functionality and High Level Architecture of XSAR

Basic Philosophies Behind XSAR

XSAR is built to be able to dynamically query large information repositories of XML documents. It is **DTD independent**, so it can be used to run a query on a CML repository, and another query on an NML repository. XSAR is a **dynamic** agent, meaning it does not use an underlying database or directory scheme, but rather dynamically queries the repository on behalf of a user. There is no need for the repository site to have any special software or hardware; the only assumption is that it is a site that contains a mixture of HTML and XML pages.

XSAR does require the user to be aware of the underlying DTD, insofar as being aware of which fields the user needs searched. Essentially, XSAR reformulates the query as an XQL expression, and then launches a spider that traverses the information repository and executes the XQL query against each XML page found in the repository.

The conceptual picture of how XSAR is utilized is shown in figure 3.

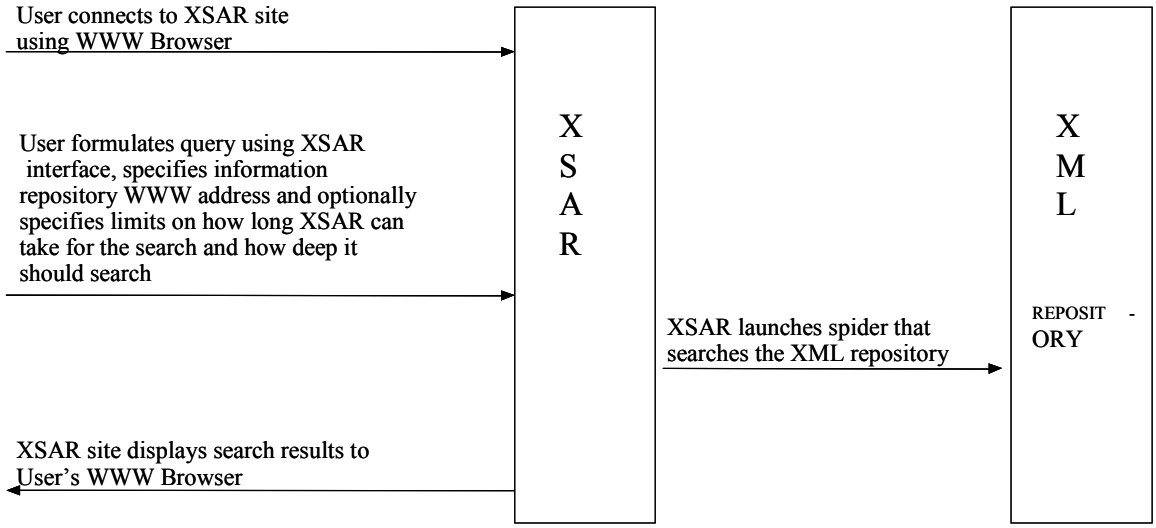

Figure 3. Conceptual Operation of XSAR

High-Level Architecture of XSAR

XSAR was built entirely using the Java Server Pages (JSP) architecture, which is part of the J2EE (Java 2 Enterprise Edition) specification, accessible currently at http://java.sun.com. This architecture has the advantage of cross platform portability, in that the agent software is operating system independent, and the search results produced (JSP pages) are browser independent. Figure 4 illustrates the high-level architecture of XSAR as per the Model View Controller (MVC) model [8].

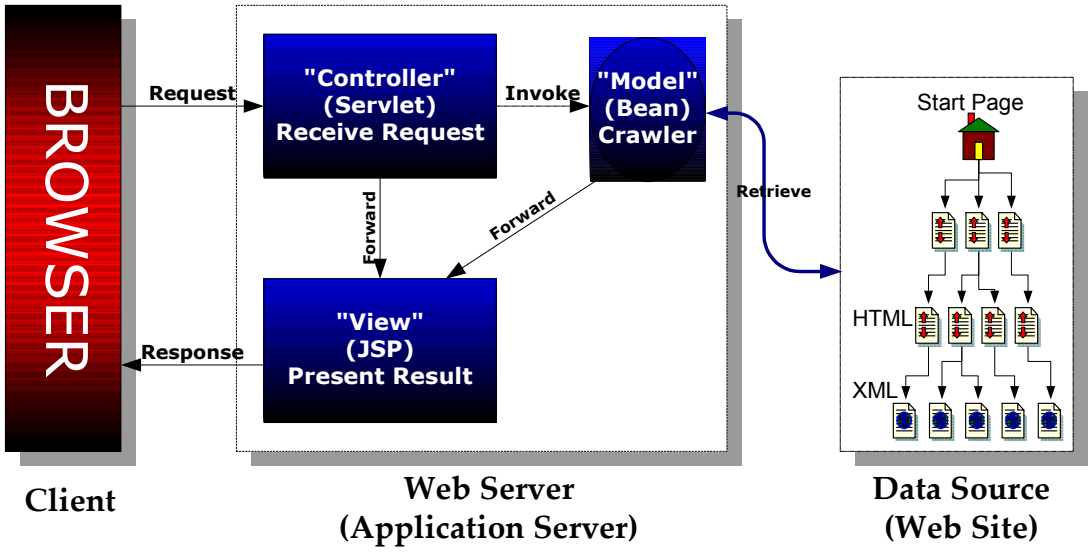

Figure 4. High Level MVC Architecture of XSAR

Choice of XML Query Language

Currently, no query languages for XML are standardized by WWW Consortium (W3C). However, several query languages for XML documents have already been proposed. For example *XQL*, *XML-QL* and *Quilt* (all accessible via http://www.w3c.org). We

selected XQL because a) its grammar is based on *XPath*[17],which has already been standardized by W3C, and b) Application Programming Interfaces (API) for XQL are available in Java.

It is important to note first, that XQL is used to query an individual XML page, and to return results from that page. It does not query **multiple** XML pages, and return combined results from them. XSAR essentially crawls through the repository, and runs the XQL query off each XML page it finds, and then combines and displays the results to the user. Second, while XSAR currently uses XQL, it's can be easily modified to use another API, if the W3C standard changes. Third, XSAR allows the usage of XQL to be transparent to the user, by providing the option of easy-to-use boxes (see Appendix 1, figure 1) to formulate the query.

Functionality of XSAR

Based on the conceptual description of XSAR (see figure 3), detailed screenshots of actual XSAR usage are shown in Appendix 1. The case in the appendix shows a search for a specific employee whose name is "Kent Smith" and whose age is more than 30. The user has a choice of specifying the search as either an XQL string ('expert mode') or formulating it using the drop-down boxes ('novice mode'). In this case, "novice mode" is used for specifying the search condition. Single quotes are used to surround string parameters. If a parameter is a number, no quotation is required. Each comparison operator has two types: case sensitive and case insensitive. Figure 1 in Appendix 1 shows the search conditions for this case. Note how the interface requires the user to be DTD aware (they need to specify the element or tag whose values they want searched). One of the operators the user can select is the *contains* operator, which allows for free text search within an XML tag. However, the user does not need to be XQL aware, if entering queries in 'novice mode'.

Figure 2 in Appendix 1 shows the list of search result. In this case, only one XML documents is found. Figure 3 in Appendix 1 shows the result for choosing "See Matches" in figure 2. Only the matched part of *department2.xml* is displayed. Figure 4 in Appendix 1 shows the result for choosing "See whole XML" in Figure 2, Appendix 1. The entire *department2.xml* is displayed. Having described the functionality of XSAR, we next describe the low-level architecture of its components.

Low-Level Architecture of XSAR

As illustrated in figure 5, the low-level architecture of XSAR follows the MVC model. *XMLRetriever* and *ShowMatch* are the Controller: they are Java Servlets which are responsible for receiving the requests from the client. The *Agent* component takes responsibility for retrieving the data from the website (Model). The search results are presented as JSP pages (View).

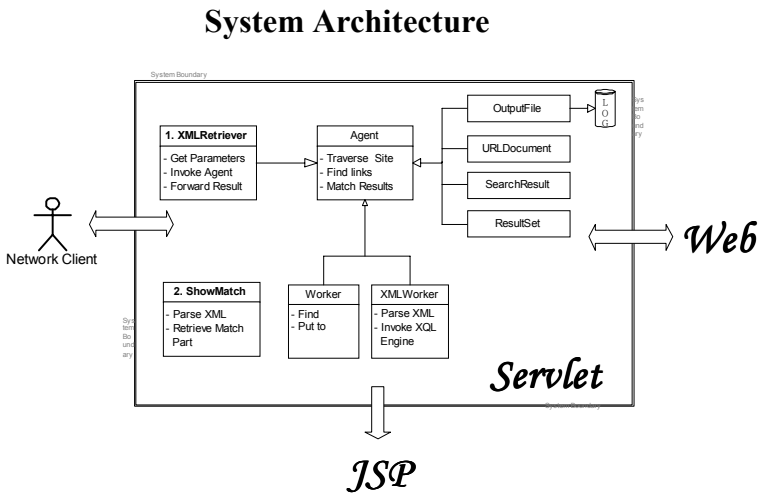

Figure 5. Low Level Architecture of XSAR

Once *XMLRetriever* receives the request from the client, it generates the XML query string and hands it to *Agent*. *Agent* essentially crawls through the target repository, utilizing a breadth-first search algorithm. It parses the starting page (specified by the user)

and identifies links. Each linked page is then parsed (if it is an HTML page) or parsed and searched (if it is an XML page). *ShowMatch* gets the search results from the agent program and puts them into JSP files for presenting.

XSAR calls on the following APIs: a) *Xerces* (http://xml.apache.org/xerces-j/) for Java for XML parsing; b) *GMD-IPSI XQL Engine* (http://xml.darmstadt.gmd.de/xql/) for XQL querying and c) *Tomcat* (http://jakarta.apache.org) as the underlying Servlet and JSP Container.

The source code underlying XSAR is available at its website. Next, we describe the performance considerations associated with the design of XSAR, and compare qualitatively the performance of a search agent with alternate methods of searching.

Performance Considerations

We discuss the performance of XSAR using the four metrics defined in section 1: *recall*, *precision*, *percentage of dead links* and *response time*. We compare XSAR to three alternatives: a) **keyword based search engines** that employ a central database of a universe of pages; b) **directory based search engines** that contain links to a universe of pages; and c) **information repositories that use an underlying database** to store the information in (say) relational form, translate this stored information to and from XML format and offer dynamic querying capabilities using an on-site search supported by technologies like CGI, ASP or Java Servlets. As discussed in section 1, the performance of all these mechanisms is also strongly affected by how narrowly the search is specified by the user. For the purposes of the discussion below, we assume that the search specificity is held constant.

Recall

For keyword based and directory based search engines, recall is usually dependent on the percentage of existing pages that are captured in their respective universe of pages. As the number of real-world pages on scattered WWW sites increases, it is widely accepted that the recall of these two search mechanisms will diminish. This has led to a surge of meta-search engines (*e.g.*, http://www.dogpile.com), that now attempt to combine results from multiple search engines, with a view towards increasing recall.

For information repositories that use an underlying database, the recall of a search is determined by the goodness of translation of the user's query into the underlying database language (usually Structured Query Language(SQL)) query. If the query translation is accurate, and the underlying database is searched completely, the recall should be 100%.

For agents like XSAR, the recall is dependent on whether a path exists from the specified start point to all the XML pages that exist in the repository. If this condition is true, then the recall should be 100%.

Precision

For keyword search engines, precision is determined by a) the features offered to the user to specify a query narrowly (*e.g.,* concatenated strings, wild card characters, Boolean operators); b) a design decision on the extent to which precision should be sacrificed for response time; and c) the extent of information captured in the underlying database about the real pages. All else being equal, a search engine that offers greater features for search specification should usually have greater precision, as will one where response time is put secondary to precision. For c), a search engine that only captures the title of each page will have less precision than one that captures the title and the META tags of each page.

For directory based search engines, the precision is determined by the accuracy of the directory structure in reflecting the user's world. Thus, if the directory listing stops at "motorsport clubs" but the user is only interested in "motorcycle racing" then the precision is likely to be poorer than for another directory structure that specifically contains a "motorcycle racing" directory entry. As the semantic space of web documents moves to encompass the entire world, it becomes harder and harder for a directory structure to be all things to all people. Hence the precision of directory based search engines is expected to decline.

The precision of an information repository that uses an underlying database would depend on (a) the features provided by the query interface that would allow the user to specify a query; and b) the maximum precision allowed by the underlying query language of the database. If a relational database is used, then the underlying query language would be SQL, which is widely

accepted to provide excellent precision, and the precision would depend on which features of SQL are supported in the query interface provided to the user.

For XSAR, the query interface is fully supportive of XQL. Thus, the precision of XSAR is identical to the precision of XQL, which supports *XPath* (http://www.w3c.org), the W3C standard for precision.

Percentage of Dead Links

For both keyword and directory based search engines, the percentage of dead links is determined by the actual continued existence of the pages that are listed in their universe of pages. For both these types of search engines, as the number of actual pages increases, and the size of the universe of pages goes up, the percentage of dead links would go up.

The percentage of dead links for information repositories with underlying databases would be zero, since the database *is* the information. Similarly, for XSAR, since the search is conducted on the real repository, the percentage of dead links is zero.

Response Time[3]

The response time of keyword based search engines is dependent on a) the algorithm used to perform the key word match with the database that has the universe of pages and b) the extent to which the designers are willing to sacrifice precision for response time.

The response time for directory based search engines is dependent on the ease of navigation of the directory structure. As the semantic complexity of the directory structure increases, response time becomes worse.

For information repositories with underlying databases, the response time depends on the complexity of the query specified and how well the database is tuned for that query. Thus, for a relational database, a range query on non-indexed columns with several joins is likely to take significantly longer than a simple SELECT query. Thus, these repositories would need a database administrator to keep track of most frequent queries, and ensure proper tuning.

For agents like XSAR, response time is determined by the time taken by the crawler to crawl through the target repository. This is dependent on exogenous factors like network speed, performance of the web server of the target repository and the size of the target repository. In designing XSAR, we used three endogenous strategies to minimize this time, for given values of exogenous factors: a) A multi-threaded agent; b) A proxy server for caching and c) allowing the user to specify response time threshold. We next describe each of these strategies.

As the agent program fetches new pages from the target repository, it spawns threads to parse these fetched pages. HTML pages are only parsed for links, while XML pages are parsed and searched for the query. One design criterion here is that spawning a new thread can become expensive if the pages being parsed and/or searched are small. After experimenting with different real XML repositories (see Appendix 2), we decided to use five threads per search. Since XSAR is an open source software, this can be modified under different installations.

For proxy caching in XSAR, we used the DeleGate proxy server, developed by Yutaka Sato from Electro Technical Libratory in Agency of Industrial Science and Technology, MITI Japan. A proxy server is useful in reducing network traffic, since it caches frequently accessed pages, and hence reduces an agents (or browsers) requests to a distant WWW server. We tested the performance of using the proxy server for a set of cases. The results of using the proxy server are shown in Appendix 2

Finally, XSAR allows the user to set the threshold response time and maximum depth that the agent should search the target repository. This allows XSAR to be used by different classes of users, *e.g.,* one class of users may want to search large repositories comprehensively, and leave the agent running overnight, while another class may want quicker responses for more shallow searches.

[3]We assume a given size of the universe of pages in this section.

We tested XSAR for three cases, described in Appendix 2. We found that using a multi threaded program had the maximum impact in reducing response time, with a proxy cache playing a secondary role. The results (see Appendix 2) indicate that XSAR is clearly slower than the other mechanisms available for searching, which is to be expected since it searches a target repository in real time. Using the three strategies described above causes the search time to be under one minute for real world repositories that are still fairly small. Since XSAR is unique, in that it searches repositories in real time, we anticipate its usage to be by users who are not looking for sub-second responses (as is typical in casual browsing), but are rather looking for an agent to conduct a more serious search on their behalf on a large repository and are anticipating response times in minutes or hours.

Figure 6 below summarizes the performance comparison between the four search mechanisms. A ↓ implies that the metric is expected to become lower over time.

Metrics → **Search Methods↓**	**Recall**	**Precision**	**Percentage Dead Links**	**Response Time**
Keyword based Search Engine	Low ↓	Average	High ↑	Least
Directory Based Search Engine	Low ↓	Low ↓	High ↑	Average
Information Repository using underlying database	100%	High	None	Least
XSAR	100%	Highest	None	Most

Figure 6. Summary of Performance Metrics for the Different Search Mechanisms

Based on the discussion above, it is clear that information repositories of the future can either store information in an underlying database, or use a dynamic search agent like XSAR. While the performance metrics for an underlying database are better than for a dynamic search agent, there are some application maintenance issues that are worth considering. For illustration purposes, let us assume a large news organization called *rabbitnews* has decided to store all its news articles using the NML DTD.

If an underlying database is used, *rabbitnews* will have to write application code for a) converting information from NML to the underlying database schema, b) extracting information from the underlying database and presenting it as an NML document and c) translating queries formulated in a standard XML query language like XQL to the underlying query language of the database like SQL. Furthermore, as the number of articles stored by *rabbitnews* increases, it may need to use multiple databases for storage, and to coordinate the application code in a), b) and c) mentioned above, across the multiple databases. It will also need to hire at least one database administrator to maintain this information, and tune the database for queries, as discussed earlier in this section.

On the other hand, if the news articles are stored as multiple XML pages, and accessible using XSAR, the application code required from the perspective of *rabbitnews* will be an editor to create an NML page. Since NML is a standard, there may be potentially several NML editors available freely. Thus this storage mechanism is more desirable from an application maintenance perspective. The scalability of this mechanism is dependent on a) the ability of *rabbitnews* to serve out pages from a large selection of static pages, a technology that is well understood, and b) the ability to provide quick content-based query results, using an agent like XSAR.

The above example illustrates the pros and cons of each search mechanism.

Conclusion and Future Research

In this work we analyze possible search mechanisms for large XML repositories, and present XSAR, a dynamic, DTD independent search agent. We describe its functionality and its architecture, analyze its performance and compare it to other search mechanisms. XSAR is free software, available under the GNU public License. It can be accessed at http://bajaj.heinz.cmu.edu/XSAR/Xsar/, where users can use it to search XML repositories on the WWW. For future work, we plan on a) expanding XSAR to search multiple repositories in parallel, and collate the results, b) explore a peer-to-peer

architecture for XSAR, wherein each repository would host its own agent and c) explore new ways to improve the performance of XSAR. We hope to attract a body of programmers to work on the XSAR project, and provide collective solutions towards searching XML repositories.

Acknowledgement

We acknowledge the insightful comments of Professor Ramayya Krishnan, of the H. John Heinz III School of Public Policy and Management, Carnegie Mellon University, in the development of XSAR, and the writing of this paper

References

[1] R. Khare and A. Rifkin, "XML: A door to automated web applications," *IEEE Internet Computing*, vol. 1, pp. 78-87, 1997.
[2] T. Berners-Lee, J. Hendler, and O. Lassila, "The Semantic Web," in *Scientific American*, vol. September, 2001.
[3] N. J. Belkin and W. B. Croft, "Retrieval Techniques," *Annual Review of Information Science and Technology*, vol. 22, pp. 109-145, 1987.
[4] D. Billsus and M. Pazzani, "Learning Collaborative Information Filters," presented at Machine Learning: Proceedings of the Fifteenth International Conference, 1998.
[5] B. B. Anderson, A. Bajaj, and W. Gorr, "An Estimation of the Relative Effects of External Software Quality Factors on Senior IS Managers' Evaluation of Computing Architectures," *Journal of Systems and Software*, To Appear.
[6] F. Menczer, "Life-like agents: Internalizing local cues for reinforcement learning and evolution," in *Computer Science and Engineering*. San Diego: University of California, 1998.
[7] K. Sripanidkulchai, "The popularity of Gnutella queries and its implications on scalability," . White Paper, SCS, Carnegie Mellon University, Pittsburgh, 2001.
[8] M. Foley and M. McCulley, *JFC Unleashed*: SAMS Publishing, 1998.

Appendix 1
Illustration of Usage of XSAR

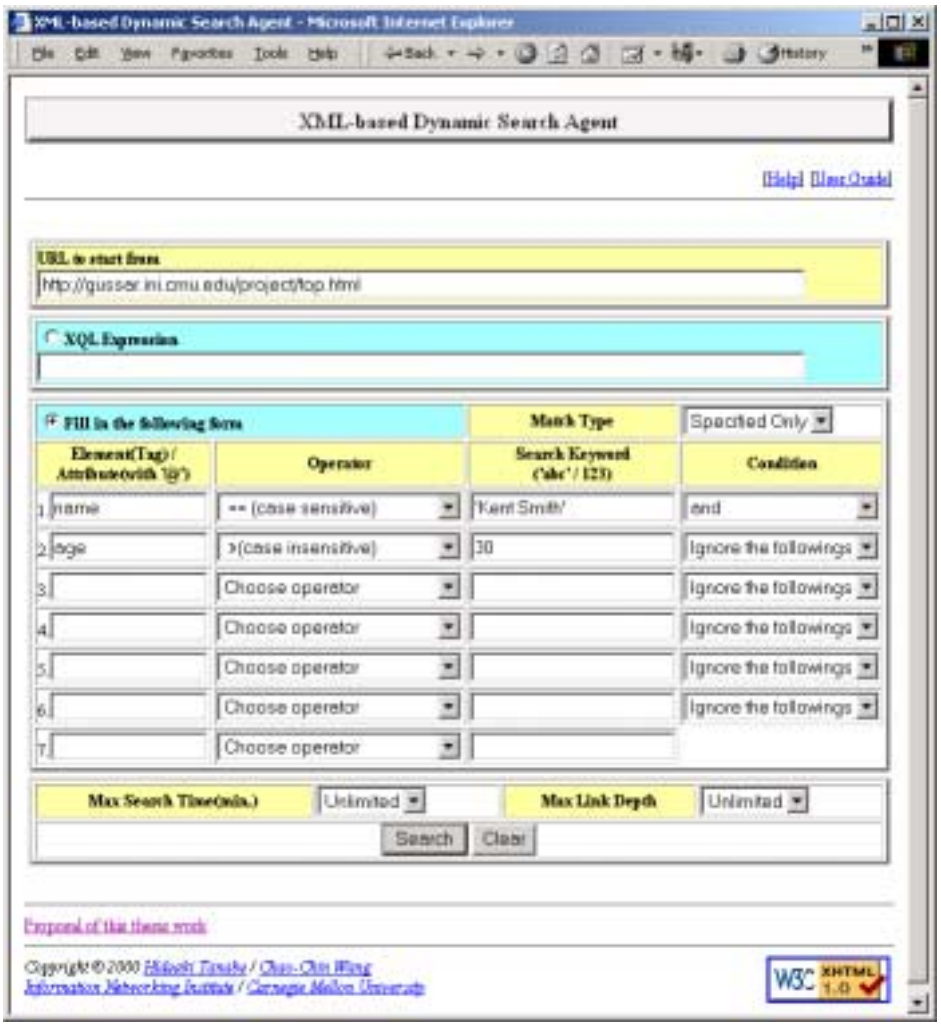

Figure A1. Creating Query For Local Test Website

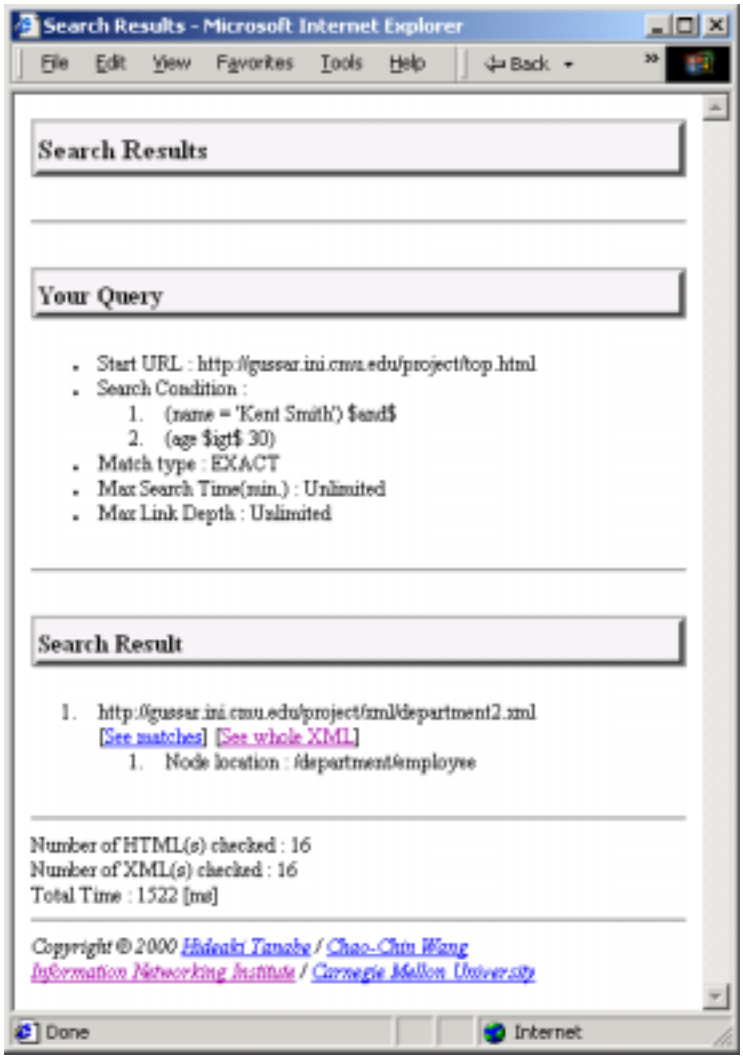

Figure A2. Search Result Page for Local Test

Figure A3. Result of "See Matches" for Local Test Website

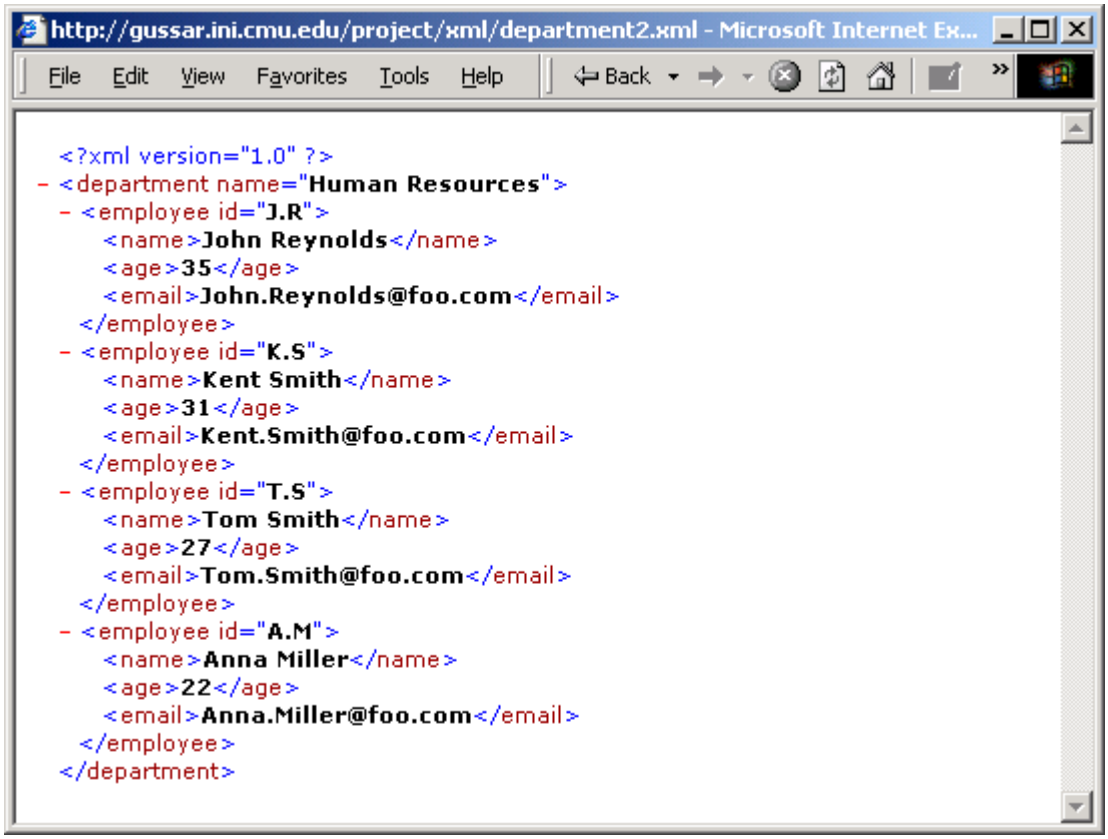

Figure A4. Result of "See Whole XML" for Local Test Website

Appendix 2
Performance Results of Test Cases

We present the results of three test cases that illustrate the performance of XSAR.

The first case was a repository created inside the same network as XSAR. Therefore, the target website and the agent server were connected by Ethernet, 10Mbps. The contents of the repository were small HTML files and XML files, less than 4Kbytes each.

The second case was a real XML repository (http://www.xml-cml.org/), which has the conclusive resource for Chemical Markup Language which is used to exchange chemical information and data via the Internet.

The third case was also a real-world case: the Dublin Core Metadata site (http://purl.org/dc/) which is a specification for describing library materials.

Table 1 shows the search time for the cases mentioned above. The search time results clearly depend on exogenous variables like network conditions, PC hardware specifications, and load status. In order to eliminate at least some variation in these factors, we tested each case five times, and present the mean values in Table 1.

Table B1. Comparison Among Experiment Cases

	Local experimental Environment	**CML**	**Dublin Core**
Number/ Average size of HTML	16 / 458 [bytes]	1 / 10490 [bytes]	34 / 16966 [bytes]
Number / Average size of XML	16 / 2756 [bytes]	173 / 10314 [bytes]	27 / 974 [bytes]
Direct access	2545[ms]	19291[ms]	44637[ms]
Via cache server	1772[ms]	14577[ms]	40305[ms]
Ratio of (w/ Cache) / (w/o Cache)	0.696	0.756	0.903