RC22716 (W0302-018) February 5, 2003
Computer Science

IBM Research Report

Virtual Documents in Information Search and Retrieval Systems

Yurdaer N. Doganata, Youssef Drissi, Tong-Haing Fin, Lev Kozakov
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Research Division
Almaden - Austin - Beijing - Delhi - Haifa - India - T. J. Watson - Tokyo - Zurich

Virtual Documents in Information Search and Retrieval System

Yurdaer Doganata yurdaer@us.ibm.com
Youssef Drissi youssefd@us.ibm.com
Tong-Haing Fin thfin@us.ibm.com
Lev Kozakov kozakov@us.ibm.com

IBM T.J.Watson Research Center

ABSTRACT
Traditional information search and retrieval systems crawl collections of URLs pointing to the physical locations of documents or web pages, index their content, and return a set of URLs in response to user queries. The users, however, require more flexible, customizable, reliable and effective information search and retrieval systems. This paper presents an alternative approach by adopting the concepts of 'virtual documents' and 'virtual URLs' that utilize searchable component libraries created by extracting information from documents. This approach enables the design of information search and retrieval systems with enhanced and flexible crawling, indexing, retrieving and rendering features.

Categories and Subject Descriptors
H.3.1 [**Information Storage and Retrieval**]: Content Analysis and Indexing – *Indexing methods.*
H.3.3 [**Information Storage and Retrieval**]: Information Search and Retrieval – *Information filtering, Retrieval models.*

General Terms
Management. Design.

Keywords
Crawling, Virtual Documents, Customization.

1. INTRODUCTION
Information search and retrieval (IS&R) systems deal mostly with document sets. These document sets could range from relatively small collections of files stored on a desktop computer to billions of documents distributed over the Internet. Some large-scale Web search engines [1] were built to search and retrieve hundreds of millions of web pages, while other systems focused on selecting smaller sets of documents relevant to predefined topics [2]. As the scale of information retrieval systems and applications increased, the user expectations are also shifted from just retrieving electronic copies of documents from large repository of structured content to retrieving unstructured context focused information with customizable rendering capabilities. Companies are pressured by increased customer expectations in the areas of information retrieval while motivated to provide support over the Web with reduced cost.

The design of contemporary information retrieval systems is driven by new user requirements aimed at increasing goal attainment and hence user satisfaction. These requirements can be grouped as follows:

- The user experience should not be limited to or solely determined by the content authoring process. The system should allow rendering the content differently based on different context as well as based on custom layout. This is particularly important when the same search service is used by multiple organizations globally where each organization comes forward with their own set of usability requirements.
- The content to be indexed should not be limited only to the terms used by the content authors. A pre-indexing process should enhance the content with possible text analysis applications by including the variants of important terms, synonyms, meta tags, topic and task classifications, etc. The pre-indexing processing will provide information about the content more than the content author could provide. This is realized by comparing and contrasting the document, both linguistically and statistically, to the other documents in the corpus.
- The availability of the content should not depend on the availability of the Web servers placed in front of the content. Today, almost all documents are served through a Web server. The content becomes unreachable when the server is down or cannot be reached.

In this paper, we present alternative models of fundamental operations in IS&R systems, designed to satisfy the requirements above. The rest of the paper is organized as follows. Next section defines the concepts of virtual document and URLs and describes how these concepts are used to create and index multiple views of

the same documents in different contexts and make them searchable. Section 3 introduces a new crawling model called virtual crawling that allows indexing beyond the authored content. Section 4 is dedicated to the concept of virtual rendering where the document views are built dynamically based on predefined custom layouts.

2. VIRTUAL DOCUMENTS AND VIRTUAL URLS

2.1 Shortcomings of traditional approach

Content explored by most IS&R systems usually comprises static HTML pages and other static documents that can be accessed through the Internet or corporate Intranets. In order to serve its customers, the IS&R system creates full textual index [1] of all the documents, using static document URLs as the unique identifiers.

However, this traditional approach has a number of shortcomings, especially when it is applied to enterprise document collections. The size of such a collection may be relatively small, if compared to the whole Web, but a company may introduce special requirements that are not addressed by this approach. In particular, the company may impose certain standards on document layouts, or establish complicated hierarchical classification of document types. Such standards may evolve with time, so multiple enterprise departments will constantly need to update their documents. Another problem is that some departments may want to customize the view of documents they present, or to present different views of the same document in different contexts. All this issues can hardly be addressed by the traditional IS&R system.

2.2 Searchable component library

In this paper we employ the concept of 'virtual documents' and 'virtual URLs' to make IS&R systems really flexible in crawling, indexing, retrieving and rendering the content. The concept of virtual documents, used in this paper, in fact, is a limited version of the original definition given by [16] and others [17, 18]. The original definition assumes that the content elements of a virtual document might be built on-the-fly from several data sources, e.g. as a result of SQL query submitted to several databases. Our version of the concept assumes that the content elements of virtual documents along with the associated metadata are already created and stored in the repository, so when the virtual document is retrieved, the content elements may be put together to build appropriate document view on-demand.

This concept is realized using the Searchable document component library that stores documents as sets of content elements or components along with the metadata. To create the component library the documents are extracted from their original repositories, and passed through a document migration pipe to the new centralized repository. This migration pipe comprises the following modules:

- Document decomposer that breaks the document content into content elements.
- Document processor that extracts metadata and stores content elements along with the metadata in the component library. The Document processor may use advanced text analysis tools to enrich the document content with automatically generated summary, or synonyms of the most salient terms that appear in the document. Eventually, the Document processor encapsulates all the document content elements in one XML document, using predefined XML schema.
- Component library repository that stores XML documents along with the associated metadata, and provides access to the content elements and the metadata. Each content element is tied to its original document, so that all the elements of a particular document can be retrieved using the given unique document ID.

After the content components are stored in the repository, the content is passed to the search engine crawler for indexing. To perform the indexing document content elements are put together by another module of the component library - Document view builder. This module is able to build a document content view from the content elements and associated metadata, and provide the document content with the unique URL, based on the document ID. The Document view builder also performs the main role in constructing the document view and/or layout when retrieving the documents from the component library.

A set of the document content elements together with the associated metadata, stored in the Searchable component library, along with the set of predefined document views, supported by the Document view builder, represents an instance of the virtual document. Each virtual document is provided with unique ID, generated when the document is stored in the library. The document ID is used to index and retrieve the document content elements, and build the document view. This ID along with the address of the component library access point and customization parameters represents the virtual URL of the document. In the next section we will show that virtual URL enables flexible URL-based or parameter-based filtering of search results.

3. VIRTUAL CRAWLING

3.1 Traditional Crawling Methods

The process of collecting documents, usually distributed over a large computer network or stored on a stand-alone system, is often called crawling. Preparing the content for crawling can include specific document preprocessing to be completed before the indexing phase. For example, in local (intranet) search systems that require the indexing of different document types, there might be a need for a preprocessing that converts the documents to a unified format compatible with the search engine interface. If the same content is to be crawled by different search engines that require specific formats, the content might need to be replicated several times to have, for each search engine, a corresponding replicated content formatted according to the crawler's rules. This type of replications can also be relevant if the documents need to be presented in different contexts or with different views. More details about some of these crawling types and their shortcomings are presented in the next few sections.

3.1.1 Crawling the Web

In the context of the World Wide Web, crawlers are programs that automatically traverse the Web graph, retrieving pages and building a local repository of the portion of the Web that they visit. Depending on the application at hand, the pages in the repository are either used to build search indexes, or are subjected to various forms of analysis (e.g. text mining) [5].

Most of Web crawlers retrieve content only from the publicly indexable Web, i.e., the set of Web pages reachable purely by following hypertext links, ignoring search forms and pages that require authorization or prior registration [5]. Moreover, in spite of impressive resources including high-end multiprocessors and well crafted crawling software, the largest crawls cover only 30-40% of the Web [3,4].

3.1.2 Crawling Databases

In this scenario, the content to be searched and indexed is not organized as regular files, but rather as data records stored in a relational database. Each record or piece of information is indexed individually. At the run time, a search query is submitted against the index, and a list of matching records is returned without compiling them into "real" document. In a sense, this process disregards the relations between the different pieces of data. The Figure 1, below, illustrates this scenario.

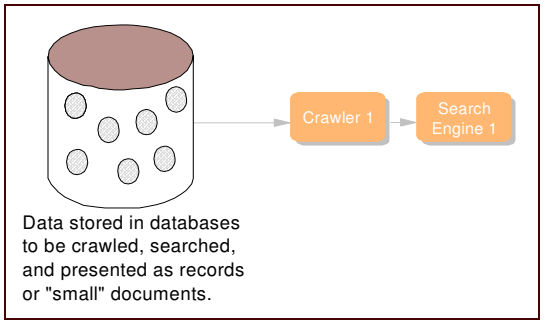

Figure 1. Crawling databases.

3.1.3 Crawlers with Proprietary Interfaces

In this scenario multiple search engines need to index the same content. But the corresponding crawlers have a proprietary interface and require a specific format for the input documents. Therefore, a preprocessing step takes place to replicate the content and convert it to the format supported by the crawler's interface. The Figure 2, below, illustrates this scenario.

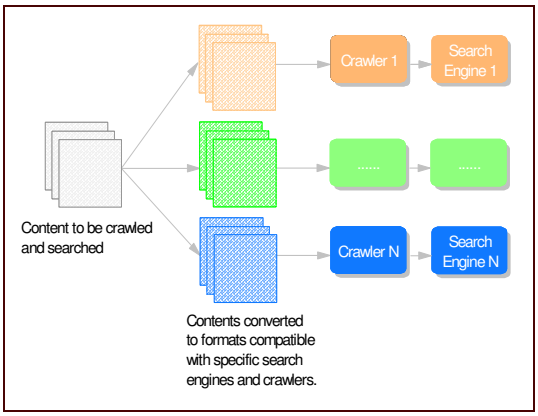

Figure 2. Crawlers with Proprietary Interfaces.

3.1.4 Crawling Multiple Views

In this scenario, every document could have multiple variants or views depending on the context. Such context could be defined by the user personalization preferences. Moreover, the search systems and services, in this case, require the indexing of all the document views and structures. One way to achieve this goal is to replicate the documents for each required view. Each replication would contain the documents converted to a specific view or transformed to a specific structure compatible with a given schema. The Figure 3, below, illustrates this scenario.

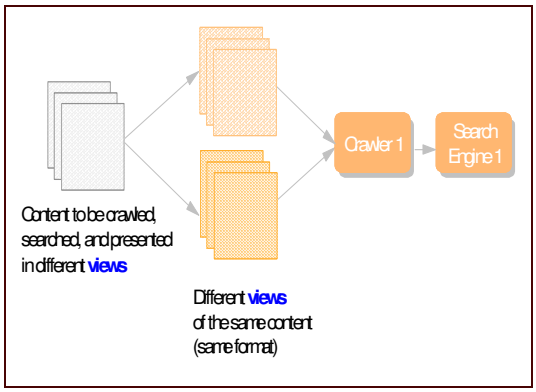

Figure 3. Crawling the same content with multiple views.

3.1.5 Focused Crawling

Generic crawlers and search engines might do poorly when it comes to fulfilling the need for highly specialized information where the user can explore his interest in depth [4,12]. The goal of a Focused Crawler [2] is to selectively seek out pages that are relevant to a pre-defined set of topics. The importance of this type of crawling comes partially from the scaling challenges posed by rapid growth of the World-Wide Web. Focused crawlers, in general, have a good recall and precision because they restrict themselves to a limited domain [13].

3.2 A Virtual Crawling System

The current crawling methods present interesting problems worth to study and solve. For instance, in the case of a content crawled by different search engines with proprietary interfaces, there is a need to replicate the content with a specific format for each search engine. This operation not only multiplies the storage volume needed by the number of search engines, but also introduces a static process to be executed every time a search engine is added. This results in a poor flexibility and automation level for the crawling process. The same problem is faced when multiple views of the same content need to be indexed. Here again, the number of views multiplies the storage volume, and the process remains difficult to adapt to the addition of a new content view. When crawling databases, the search engine indexes unprocessed records of data. The presentation of the data, hence, the user experience, is limited by the database layout. In this case, this limitation applies in addition to the ones encountered in the crawling modes mentioned above.

The "Virtual Crawling" is a crawling process where the documents are not stored as physical files, but as granular elements of the actual content. These elements are stored in a database as reusable pieces of data. A document view builder module then builds a document on demand, with the desired elements. The document view builder takes also as input a schema that describes in detail the element types to be collected and assembled, as well as the structure of the final document view. This way, it will be possible to create any document view, based on user's choice or preferences. A document viewer module renders dynamically the desired view of the content. This module, hence, is used to present the same content in different contexts. The generated documents do not have to be stored physically any more, they become "virtual documents". In a sense, there are no real physical document files in this crawling process. These virtual documents are built on demand with the desired view in a certain context, and with no need for multiple replications of physical document files. This design further allows for more flexibility in GUI without the necessity of adding a new view of the existing content. That means that not only the maintenance cost, but also the storage cost is reduced.

3.2.1 System Architecture

The Virtual Crawling architecture is illustrated in Fig.4 below. The component Extractor module extracts the documents from the original data source and carves the document components and sections, then stores them into a database. The Document View Builder is responsible for collecting context information, about the crawler's interface and the corresponding document schema, from the configuration module. After collecting all the necessary input, the Document View Builder creates the document streams in the memory and feeds them to the crawler. The configuration module maintains all the data about the context of the crawling process, such as the crawler interface, formats supported, schema, structure, and view in which the document have to be created. The Format Identification module communicates with the crawler to detect automatically the crawler requirements regarding its interface and supported document formats, as well as the formats of seed URIs to be crawled, when applicable.

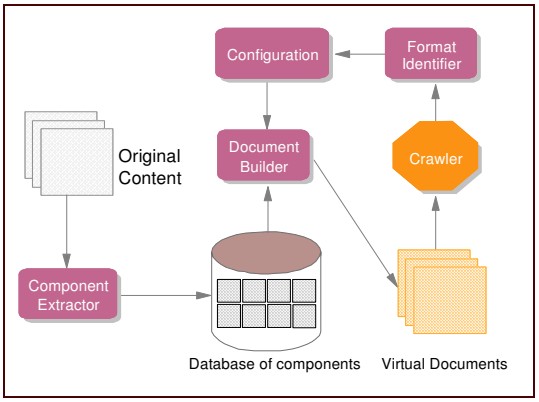

Figure 4. A Virtual Crawling System.

3.2.2 Component Extractor

The Component Extractor module (see Figure 5) is responsible for carving the documents into components that comply with a given specification compiled into an XML Schema. The documents are accessed one by one by the extractor through an access method specified by the configuration module. The documents are then passed to the document carver component that takes also as input an XML schema that specifies, in detail, how to cut the documents up, as well as the formats, sizes, and other attributes of the resulting sections and components. The final components are then stored in a database with the meta-data that preserves the relations between these components themselves and also their association with the original document.

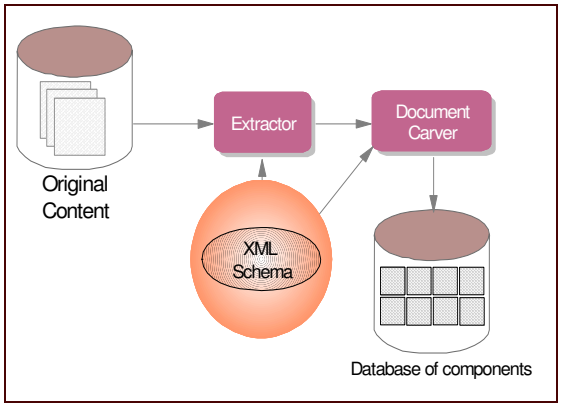

Figure 5. Components Extraction.

3.2.3 Interface Identification

The Interface identifier module (see Figure 6) is responsible for detecting the crawler's type and meta-information and sending the results to the configuration module for further processing. To achieve this goal, it establishes a communication with the crawler. Both the module and the crawler should to comply with a certain protocol standard. Otherwise, the crawler information needs to be fed manually to the configuration module. Through an established connection, the module requests the specification of the method

call(s) and procedures to be followed in order to crawl a set of documents to be indexed by the search engine. The crawler responds to that request by sending an XML file, which contains all necessary details describing the crawler's interface and the details of the supported formats.

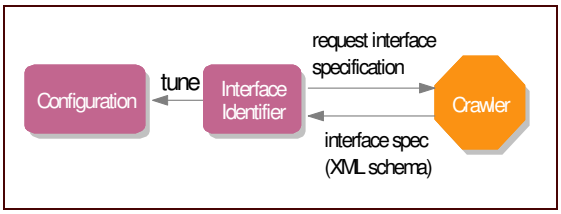

Figure 6. Interface Identification Module.

3.2.4 Document View Builder

The Document View Builder module (see Figure 4) is responsible for creating customized documents based on context and user preferences. This information comes from the configuration module that stores the data about the crawler's interface and the documents schema. After collecting all the necessary input, the Document View Builder creates the document streams in the memory and feeds them directly to the crawler. Maintaining this flow avoids the creation of physical files on the "hard drive". Once the document structure is complete and complies with the XML document schema, the document viewer builds the final version of the document that should be presented on the graphical user interface. This final view is dictated by the personalization and context information given by the configuration module.

3.2.5 Advantages of Virtual Crawling

Virtual Crawling solves many problems and limitations found in traditional crawling methods. Here are some of the advantages provided by the Virtual Crawling process:

- Virtual Crawling avoids increasing the storage requirements for replication purposes.
- Virtual Crawling enables using crawlers with different requirements on the same content.
- Virtual Crawling enables building, crawling, and indexing multiple views without duplicating or replicating the original content.
- Virtual crawling may facilitate the process of focused crawling by enabling generation of multiple context focused URLs for the same content to focus on different aspects or domains that the document might be related to.

4. VIRTUAL RENDERING

4.1 Retrieving and Rendering Documents on the Web

Most of the IS&R systems on the Web as well as corporate systems use simplistic models of document retrieval and rendering. The most popular search gateway - Google.com - just puts original document URLs in the search results, and allows opening of the original documents directly in the user's browser. Other major search portals, like AltaVista.com or AllTheWeb.com, use simple click-through model that facilitates logging of user clicks, but does not intervene with the direct displaying of the document content. Large corporate search portals, like Hewlett Packard (http://search.hp.com/), also use the click-through model, and display original documents. Others, like IBM.com, just use direct links to the original documents.

While Web search portals allow users to view 'pure' original documents in their browsers, enterprise search portals often use specific page layouts to display their documents. In many cases such a layout comprises the enterprise 'masthead' section, document specific left navigation bar, and the enterprise footer. The document content itself is displayed in the so-called 'work area' of the page. Usually, this layout is a part of the document HTML source (for HTML documents), so the same pieces of HTML code are replicated many times. When the enterprise standard layout changes, all the documents need to be updated to adopt the new layout.

4.2 Retrieving and Rendering Virtual Documents

In the previous sections of this paper we mentioned the major shortcomings of the traditional approach to the document retrieving and rendering. The general problem of the traditional model, as applied to the document retrieving and rendering, is that the document content and layout, hence the user experience, is defined by content providers, and cannot be easily customized to fit the user's context.

The concept of virtual documents and virtual URLs, as it was defined in the section 2 of this paper, helps to separate user experience from the content authoring. This concept assumes that the documents are not associated with static URLs any more, so the same content may have several different context focused URLs - one for each predefined view. For instance, if the document is supposed to be used by both general customers and entitled customers, it may have different views for different categories of customers, so there may be two different URLs associated with the same content.

The Searchable component library that facilitates the implementation of this concept, as described in the section 2 of this paper, provides special module - Document view builder - to retrieve the content and build appropriate document view on demand. One possible way of implementing the Document view builder is based on creating a set of predefined XSL tables that are applied to the XML document content, retrieved from the Searchable component library, every time a certain document view needs to be built.

4.3 Remote Site Customization

In this subsection we briefly discuss one of the practical benefits of using the concept of virtual documents and virtual URLs. We consider the corporate technical support system used by customers of different departments of a large company. Each department may want to present the search results and technical documents to their customers in a different way, adding their own ads, promotions, etc. To meet this requirement the technical support system needs to support remote site customization (RSC). The idea of RSC is rather simple: each remote site, which wants to

present the shared system content in a special format or layout, is allowed to store and register its own forms. When the system gets a request from this remote site, it will use appropriate form to build the customized view of the content.

The Searchable component library, realizing the concept of virtual documents and virtual URLs, provides a simple and effective solution for the RSC problem. Since all the documents are stored in the component library in XML form, the customization can be easily performed by using XSL tables. The system has an extendable collection of XSL tables for all supported document views or layouts. Each remote site is allowed to register its own set of XSL tables for each document view or use default XSL tables. Thus, when the request comes from registered remote site, the system will use appropriate XSL table to build the required view of selected document.

5. RELATED WORK

Web crawlers, also known as robots, spiders, worms, walkers, and wanderers, are almost as old as the web itself. The first crawler, Matthew Gray's Wanderer, was written in the spring of 1993, roughly coinciding with the first release of NCSA Mosaic. Several papers about web crawling were presented at the first two World Wide Web conferences [10,11,12].

Generic architectures of a focused crawler [2,6] can use a classifier and a distiller as the major components. The user interest is specified by a set of example pages. The text classifiers measure the relevance of the visited web pages, and the hyperlinks are initially weighted by the relevance of their destination page. The distiller supports different strategies of ordering the links at the crawl frontier.

Google's search engine uses a distributed system and multiple machines for crawling [13]. The crawler consists of different functional components running in different processes. A *URL server process* reads URLs out of a file and forwards them to multiple crawler processes. Each *crawler process* runs on a different machine, is single-threaded, and uses asynchronous I/O to fetch data from up to 300 web servers in parallel.

The Internet Archive also uses multiple machines to crawl the web [15]. Each crawler process is assigned up to 64 sites to crawl, and no site is assigned to more than one crawler. Each single-threaded crawler process reads a list of seed URLs for its assigned sites from disk into per-site queues, and then uses asynchronous I/O to fetch pages from these queues in parallel. Once a page is downloaded, the crawler extracts the links contained in it. If a link refers to the site of the page it was contained in, it is added to the appropriate site queue; otherwise it is logged to disk. Periodically, a batch process merges these logged "cross-site" URLs into the site-specific seed sets, filtering out duplicates in the process. Mercator [8] is a scalable, extensible web crawler written entirely in Java. It's designed to scale up to the entire web. It achieves scalability by implementing data structures that uses a bounded amount of memory, regardless of the size of the crawl. It's designed in a modular way, with the expectation that new functionality will be added by third parties.

In the area of extensible web crawlers, Miller and Bharat's SPHINX system [14] provides some of the same customizability features as Mercator [8]. In particular, it provides a mechanism for limiting which pages are crawled, and it allows customized document processing code to be written.

Virtual documents are not as popular as crawlers or search engines, so the number of publications in this area is significantly more limited. Interesting document interpreter system is presented by Paradis and others [19]. This system allows gathering information from multiple sources, and combining it dynamically to produce a virtual document. Wilkinson [18] presents limited bibliography on virtual documents and their usage in IS&R systems. Watters [17] discusses major research directions in this area.

6. CONCLUSION

We have employed the concept of virtual documents and virtual URLs to create alternative models for crawling, retrieving and rendering document content in IS&R systems. Our purpose is to meet new customer requirements especially in Enterprise Search Systems. This approach enabled indexing the enhanced content, rendering custom layouts, flexible URL and parameter based filtering the search results.

7. REFERENCES

[1] S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In Proceedings Of the 7th International World Wide Web Conference (WWW7), 1998.

[2] S. Chakrabarti, B. Dom, and M. ven den Berg. Focused crawling: A new approach to topic-specific web resource discovery. In Proceedings of the 8th International World Wide Web Conference (WWW8), 1999.

[3] K. Bharat and A. Broder. A technique for measuring the relative size and overlap of public web search engines. In Proceedings of the 7th World Wide Wide Web Conference (WWW7), 1998.

[4] S. Lawrence and C.L. Giles. Searching the World Wide Web. Science 280, 98-100, April 1998.

[5] S. Raghavan and H. Garcia-Molina. Crawling the hidden Web. In the Proceedings of the 27th Intl. Conf. on Very Large Databases (VLDB), pp. 129-138, September 2001.

[6] D. Gillmor. Small portals prove that size matters, Tech column in San Jose Mercury News, December 1998.

[7] Ester M., Groß M., Kriegel H.-P. Focused Web Crawling: A Generic Framework for Specifying the User Interest and for Adaptive Crawling Strategies. The 27th International Conference on Very Large Databases (VLDB '01), Rome, Italy, 2001.

[8] S. Chakrabarti, B. Dom, and M. ven den Berg. Distributed Hypertext Resource Discovery Through Examples. In Proceedings of the International Conference on Very Large Databases (VLDB'99), pp.375-386, 1999.

[9] A. Heydon and M. Najork.. Mercator: A Scalable, Extensible Web Crawler. World Wide Web, Vol. 2, nbr. 4, pp. 219-229, 1999.

[10] O. McBryan. *GENVL and WWWW: Tools for Taming the Web*. In Proceedings of the First International World Wide Web Conference, pages 79–90, 1994.

[11] D. Eichmann. *The RBSE Spider – Balancing Effective Search Against Web Load.* In Proceed-ings of the First International World Wide Web Conference, pages 113–120, 1994.

[12] B. Pinkerton. *Finding What People Want: Experiences with the WebCrawler*. In Proceedings of the Second International World Wide Web Conference, 1994.

[13] S. Brin and L. Page. *The anatomy of a large-scale hypertextual Web search engine*. In Proceedings of the Seventh International World Wide Web Conference, pages 107–117, April 1998.

[14] R. Miller and K. Bharat. *SPHINX: A framework for creating personal, site-specific Web crawlers*. In Proceedings of the Seventh International World Wide Web Conference, pages 119–130, April 1998.

[15] M. Burner. Crawling towards Eternity: Building an archive of the World Wide Web. Web Tech-niques Magazine, 2(5), May 1997.

[16] T. Gruber, "Virtual Documents", A talk given at the Stanford Computer Forum WWW Workshop - September 20-21, 1994, http://www-ksl.stanford.edu/people/gruber/virtual-documents/abstract.html

[17] C. Watters, "Information Retrieval and the Virtual Document", Journal of the American Society for Information. To appear. Hawaii International Conference on System Sciences. Maui, Hawaii. CD-ROM Publication, 1999, http://topology.eecs.umich.edu/archive/virtual_document.pdf

[18] R. Wilkinson, "User Modeling for Information Retrieval on the Web", in Proceedings of the 2nd Workshop on Adaptive Systems and User Modeling on the WWW, pp.117-119, 1999, http://wwwis.win.tue.nl/asum99/wilkinson.html

[19] F. Paradis, A.-M. Vercoustre and B. Hills, "A Virtual Document Interpreter for Reuse of Information", in Lecture Notes in Computer Science 1375, Proceedings of Electronic Publishing '98, Saint-Malo, France, pp.487-498, 1-3 April, 1998, http://citeseer.nj.nec.com/paradis98virtual.html