IDEA GROUP PUBLISHING
701 E. Chocolate Avenue, Suite 200, Hershey PA 17033, USA
Tel: 717/533-8845; Fax 717/533-8661; URL-http://www.idea-group.com
ITP4773

On the Web and Document Databases

Yangjun Chen and Ron McFadyen
Dept. Business Computing, University of Winnipeg
515 Portage Ave., Winnipeg, Manitoba, Canada R3B 2E9
{ychen2, r.mcfadyen}@uwinnipeg.ca

ABSTRACT

This paper describes the functional requirement and architecture of a software system called Web and Document Databases (WDDBs). A WDDB is a system to manage efficiently local documents and their semantic connection to remote ones. The general objective of a WDDB is to facilitate web search and internet navigation. Abstractly, a WDDB can be defined as a triple <D, U, W>, where D stands for a local document database to store XML documents structurally, U for a set of URLs with each pointing to a remote database which shares common data with the local one, and W for a Web recognizer that identifies information sources related to data items in the local database. Then, in the case that a local document database fails to answer part of a query, it is able to connect to other databases using the stored URLs to obtain data for answering the query completely. In this way, surfing of the Internet can be performed more efficiently.

INTRODUCTION

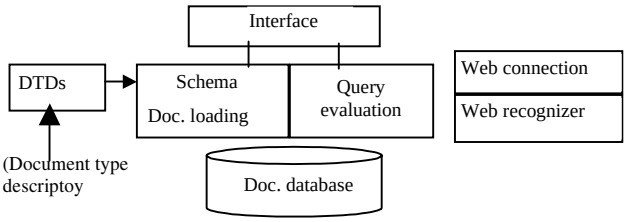
Recently, with the expansion of the Web, more and more comprehensive information repositories can be now visited easily through networks. A growing and challenging problem is how to quickly find information of interest to an individual in either a home or work setting. While navigating the Web, one may get lost in the maze of hyperlinks. A great deal of work has been done to mitigate this problem to some extent, including search engines such as *Lycos*, *AltaVista*, *Google* and *Yahoo*, web query languages such as W3QL [KS95], semistructured data management systems [Ab97, WL00] and document databases [ACC97, CA98, CA99a, CA99b, VAB96]. However, these approaches lack a general method to bring together all these aspects such as the search engine, query treatment and document management under one umbrella. In this paper, we discuss a WDDB system to provide a powerful mechanism to guide the access of information sources distributed all over the world.

Abstractly, a WDDB can be defined as a triple *<D, U, W>*, where *D* represents a local document database to store XML documents structurally, *U* represents a set of URLs with each pointing to a remote database which shares some common data with the local one, and *W* represents a Web recognizer that identifies information sources related to data items in the local database. More concretely, the remote information sources are established by storing the corresponding URLs, which are distributed over a pre-defined ontology. As an application scenario, consider a local database containing all the hotel information (*D*) in a city. Then, a query against it may get, for example, hotel prices, hotel living conditions, etc. But a user may also want to know about car rentals, sightseeing and different cuisine flavors in that city, which may be distributed in different databases. In this case, one has to switch over to those databases and submit new queries, respectively. However, if some URL links (*U*) are available and the relationships between them and the relevant local data items are specified, the system can manage to access those remote databases automatically. In addition, to obtain the URLs related to local data, a Web recognizer (*W*) is needed to explore the internet to find information sources of interest. Its other task would be to extract relevant information from the data obtained by issuing remote queries.

SYSTEM ARCHITECTURE

In terms of the discussion conducted in the introduction, we have the following system architecture for a WDDB.

Fig. 1. Architecture of a Web database



The system contains mainly three parts with each for a special functionality.

Part I - document management.

This part manages a local document database as an information source reachable over the network. Mainly, it contains:

1. A module for the schema management and the document loading. This module establishes a data schema for a given XML DTD and loads the corresponding documents into the database.
2. A module for query evaluation and
3. An interface that can be utilized for users to interact with the system.

Part II - web connection.

This part is used to connect to remote document databases distributed over the internet. For this purpose, it contains:

4. A module for web connection. In a local WDDB, a set of URLs is maintained and distributed over an ontology (see Section 4 for the definition of an ontology). That is, each concept (or a pattern) in the ontology is associated with a set of URLs pointing to remote document databases, which are related to the concept in some way. For example, for the 'car rental', we may have several URLs that are the addresses of some document databases containing the information on car rental enterprises. Therefore, a query involving a concept not available in the local resource can be sent to the associated remote document databases to get data for answering the query.

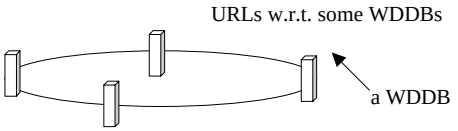
Part III - web recognizer.

The third part is used to recognize remote information sources for a given concept. It mainly contains:

5. A module for web recognition, which can be done by establishing several patterns for a concept. These patterns can be utilized to find those document databases that contain XML pages matching any of them.

From the above system architecture, it can be seen that a WDDB always works together with some other document databases distributed over the network. All the relevant document databases are considered to be semantically connected through URLs, which are associated with a concept or a pattern in some ontology defined in the local WDDB. Fig. 2 illustrates such a connection of WDDBs through URLs.

Fig. 2. Illustration of the distribution of WDDBs over network



STORAGE OF DOCUMENTS AND QUERY EVALUATION

In a WDDB, documents are stored in a document database in XML format. It may be connected to remote document databases through URLs. In this section, we mainly discuss the storage of XML documents and query evaluation in a WDDB. The discussion on database connection is postponed to the next section.

Storage of documents in a WDDB

An XML document is defined as having elements and attributes [DD94]. Elements are always marked up with tags; and an element may be associated with several attributes to identify domain-specific information. XML processors (or parsers) guarantee that XML documents stored in databases follow tagging rules prescribed in XML or conform to a DTD (Document Type Descriptor). Generally, an XML document can be represented as a tree, and node types in the tree are of three kinds: Element, Attribute and Text. These node types are equivalent to the node types in XSL [W3C98b] data model. There are some other less important node types such as comments, processing instructions, etc. The treatment of those node types is trivial and thus will not be discussed here.

- Node type of Element has an element name as the label. Each Element node has zero or more child nodes. The type of each child node is of one of the three types (Element, Attribute and Text).
- Nodes of type Attribute have an attribute name and an attribute value as a label. Attribute nodes have no child nodes. If there are multiple appearances of attributes, the order of the attributes will be ignored since the attribute order is normally not important for the document treatment.
- Nodes of type Text have strings as labels. Text nodes have no child nodes.

In Fig. 3(b), we show the tree structure representing the XML document shown in Fig. 3(a).

In Fig. 3(b), "#PCDATA" represents a data type which is more or less comparable to strings, used to accommodate text data.

To store documents in databases efficiently, the policies shown below should be followed:

- (*DTD independent*) Database schemas to store XML documents should not depend on DTDs or element types. Any XML document can be manipulated, based on the predefined relations.
- (*no loss of structural information*) The structure of a document stored in the database should be implemented in some way and can be manipulated.
- (*easy maintenance*) The cost of the main ten ace of the document structure should be kept minimum. Any update to a document will not cause the storage changes of other documents.

To reach above goals, we decompose a document into a set of elements and distribute them over three relations named: Element, Text and Attribute, respectively.

The relation Element has the following structure:

{DocID: <integer>, ID: <integer>, Ename: <string>, parentID: <integer>}.

Where DocID represents the document identifier, ID represents the element identifier, Ename is the element name (or tag name) and parentID is the pointer to the element's parent.

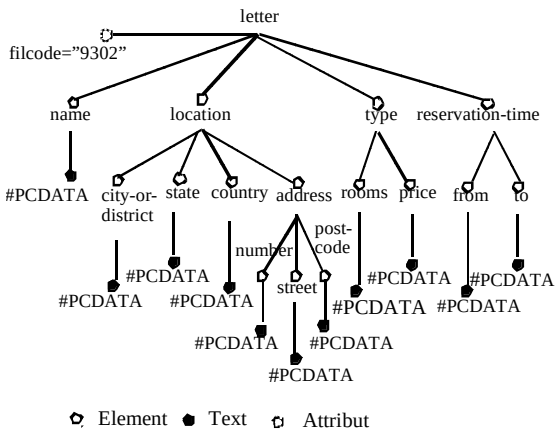
For example, the document given in Fig. 3(a) may be stored in this table as shown below.

Fig. 3. A simple document and its tree representation (a. and b.)

(a)

```
<hotel-room-reservation filecod="1302">
    <name>Travel-lodeg</name>
    <location>
          <city-or-district>Winnipeg</city-or-district>
          <state>Manitoba</state>
          <country>Canada</country>
          <address>
                <number>500</number>
                <street>Portage Ave.</street>
                <post-code>R3B 2E9</post-code>
          </address>
     </location>
     <type>
          <rooms>one-bed-room</room>
          <price>$119.00</price>
     </type>
     <reservation-time>
          <from>April 20, 2002</from>
          <to>April 28, 2002</to>
     </reservation-time>
</hotel-room-reservation>
```

(b)



From the following table, we can see that the tree structure is implemented through the "parentID" in the relation "Element", which contains pointers from child nodes to their parent. Together with the technique of path signatures to be discussed in the next section, it is especially effective for the evaluation of path-oriented queries since it is quite often to check a path bottom-up after a signature matching succeeds. But it suffers from a serious performance problem when a top-down search is desired. This can be solved as follows.

We traverse a tree structure *T* in *preorder*. Then, each node *v* in *T* will obtain a number *pre*(*v*) to record the order in which the nodes of the tree are visited. In a similar way, by traversing *T* in *postorder*, each node *v* will get another number *post*(*v*). These two numbers can be used to characterize the ancestor-descendant relationships of nodes as shown below.

Proposition 1. Let *v* and *v*' be two nodes of a tree *T*. Then, *v*' is a descendant of *v* iff *pre*(*v*') > *pre*(*v*) and *post*(*v*') < *post*(*v*).

Proof. See [Kn73].

If *v*' is a descendant of *v*, then we know that *pre*(*v*') > *pre*(*v*) according to the preorder search. Now we assume that *post*(*v*') > *post*(*v*). Then, according to the postorder search, either *v*' is in some subtree on the right side of *v*, or *v* is in the subtree rooted at *v*', which contradicts the fact that *v*' is a descendant of *v*. Therefore, *post*(*v*') must be less than *post*(*v*).

The following example helps for illustration.

Element:

DocID	ID	Ename	parentID
1	1	Hotel-room-reservation	*
1	2	Name	1
1	3	Location	1
1	4	City-or-district	3
1	5	State	3
1	6	Country	3
1	7	Address	3
1	8	Number	7
1	9	Street	7
1	10	Post-code	7
1	11	Type	1
1	12	Rooms	11
1	13	Price	11
1	14	Reservation-time	1
1	15	From	14
1	16	To	14

Example 1. See the pairs associated with the nodes of the directed tree shown in Fig. 4. The first element of each pair is the preorder number of the corresponding node and the second is its postorder number. Using such labels, the ancestor-descendant relationships of nodes can be easily checked.

For instance, by checking the label associated with *b* against the label for *f*, we know that *b* is an ancestor of *f* in terms of Proposition 1. We can also see that since the pairs associated with *g* and *c* do not satisfy the condition given in Proposition 1, *g* must not be an ancestor of *c* and *vice versa*.

To identify the parent-child relation, we associate each node with a level number. The root has the level number 0. All the children of the root have the level number 1, and so on. Then, if node *x* is the ancestor of *y* and at the same time $l(x) = l(y) - 1$ ($l(x)$ stands for the level number of *x*), we know that *x* is the parent of *y*. According the above analysis, if the top-down search of a document tree is necessary, we extend the relation schema for Element to the following form:

{DocID: <integer>, ID: <integer>, Ename: <string>, parentID: <integer>, Level: <integer>, Pre: <integer>, Post: <integer>},

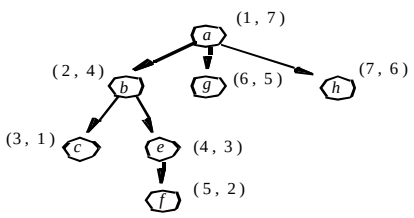
where Pre and Post are use to accommodate the preorder and postorder numbers, respectively; and Level is used for level numbers.

The relation Text has a more simple structure:

{DocID: <integer>, parentID: <integer>, value: <string>},

where "parentID" is for the identifiers of elements (stored in relation "Element") which have the corresponding text values in the original document. One should notice that a text takes always an element as the parent node. See the table for illustration.

Fig. 4. Labeling a tree



Text:

DocID	parentID	value
1	2	Travel-lodge
1	4	Winnipeg
1	5	Manitoba
1	6	Canada
1	8	500
1	9	Portage Ave.
1	10	R3B 2E9
1	12	One-bed-room
1	13	$119.00
1	15	April 20, 2002
1	16	April 28, 2002

The relation Attribute has the following structure:

{DocID: <integer>, parentID: <integer>, att-name: <string>, att-value: <string>}.

As with the relation "Text", "parentID" is for the identifiers of elements (stored in relation "Element"), in which the corresponding attribute appears. The following table helps for a better understanding.

Attribute:

DocID	parentID	Att-name	Att-value
1	1	Filecode	1302

Query evaluation in a WDDB

In a WDDB, a query evaluation comprises two processes: a local query evaluation and a remote query evaluation. When a query is submitted to the system, it will be determined which part can be handled locally and which part has to be evaluated through the internet access. As an example, consider the following query in YATL format [CCS00]:

```
Query
    MAKE
        result
            [ *hotel-and-car-rental
            [ hotel[$s], car-rental[$t]]
    MATCH docDB WITH
        hotel-room-reservation
            [ *name[$s]
                [location[$x]
                [type[$y] ] ...
        car-rental
            [ *company[$t]
                [location[$x]
                [car-type[$z] ] ... ]
     WHERE $x = 'Winnipeg' and $y = 'non smoking'
```

The above YATL query consists of a MAKE, a MATCH and a WHERE clause. The MATCH clause creates variable bindings by performing pattern matching. Hence, it contains a textual representation of a tree pattern with labeled nodes and edges. The node labels are built by XML elements (denoted by the element label, such as 'name', 'location', 'type', and so on), XML attributes (denoted by a @) and variables (denoted by a '$'). If a subpattern should occur several times, the incident edge is labeled with a '*'. The WHERE clause filters the matching results and the MAKE clause reconstructs the results according to the structure specified in it.

The tree pattern of the above MATCH clause describes a <hotel-room-reservation> and a <car-rental> element. The former contains several <name> elements, a <location> and a <type> element. The latter contains several <company> elements, a <location> and a <car-type>. Then, by evaluating this query, we'll get some hotel names located in Winnipeg, which have non-smoking rooms; and some car rental enterprises in Winnipeg as well as the car types that can be rented.

Assume that the local document database can answer the first part of the query. That is, it can provide the information on hotel room reservations, but fail to inform on car rentals. In this case, the system will send the second part of the query to the document databases pointed to by some URLs, which contain the information on the 'car rental'. If one of the remote document databases is able to evaluate the query on car rentals, the answer will be sent back to the local WDDB, contributing to a complete answer to the original query.

A remote query can be of the form: <URL><query>. For instance, assume that there is a remote WDDB with the URL: http:// www.uwinnipeg.ca/docDB, which contains the data on car-rental. The local database will issue a request of the following form to get the second part of the answer to the above query.

http://www.uwinnipeg.ca/docDB?name=$t?location=$x?car-type=$z.

The problem is how to determine where to send a remote query. In the next section, we will discuss how a local WDDB becomes aware of other document databases and know what they have.

WEB CONNECTION AND WEB RECOGNIZER

In this section, we mainly address the database connection. First, we discuss how the URLs are organized in a local database in 4.1. Then, in 4.2, we discuss how a remote information source is recognized.

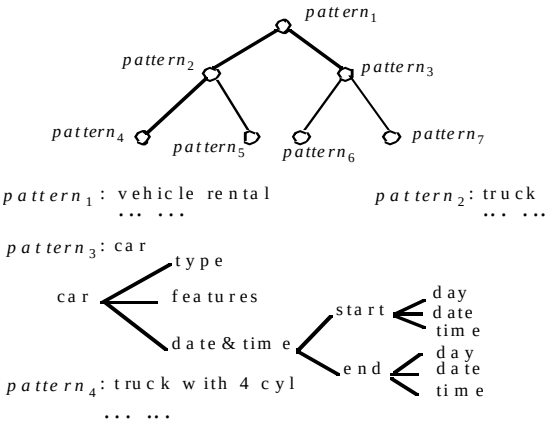
Web connection

As mentioned in the previous section, to evaluate a remote query, a WDDB has to know where to send that query. This can be done by maintaining a so-called *association list of concepts*. Each item of an association list is a triple of the form: (*G, C, S*), where *G* represents an information unit, e.g., some hotel information in a city, *C* stands for a set of URLs connecting to some remote databases containing the relevant information such as car rental in that city, and *S* is a descriptor of the relationship between *G* and *C*. Assume that the above query contains only 'hotel' part and then the local document database can not answer the query completely. However, using the corresponding item, say ('hotel', {url_1, url_2, ..., url_i}, 'car rental') in the association list, the system can switch over to the document databases pointed to by url_1, url_2, ..., url_i to obtain the information on the car rental.

In an association list, a same concept may appear multiple times and some concepts are possibly closely related. To handle these issues, we should organize the association list in a different way. To this end, we extend the concept of *mediators* discussed in [Wi92], which is originally proposed to integrate heterogeneous information sources. Concretely, A mediator is composed of two parts: an *ontology* and a set of *articulations*. An ontology is a pair (T, $\preceq$), where T is a set of names, or terms, and $\preceq$ is a subsumption relation over T, i.e., a reflexive and transitive relation over T. If a and b are two terms of T, we say that a is subsumed by b if $a \preceq b$; e.g., Database $\preceq$ Informatics, Canaries $\preceq$ Birds. An articulation is a set of relationships between the terms of the mediator and the terms of a local source. Through the articulations, the heterogeneity of local databases is suppressed.

For our purpose, a mediator in a WDDB is defined to be a *tree structure* and a set of *URLs*. In the tree structure $\Im$, a node v is a pattern that is used to identify relevant information sources. Similar to T, an edge from c to d in $\Im$ represents that the concept represented by c subsumes the concept by d. Associated with v (a node in $\Im$,), we have a set of URLs pointing to the web pages matching the pattern represented by v. As an example, consider the tree structure shown in Fig. 5.

Fig. 5. Extended ontology



Such a tree structure is called an *extended ontology* (EO for short) in the sense that a term in an ontology is extended to a more complex structure, i.e., a pattern to describe the concept more exactly. In an EO, a pattern is normally a tree to represent an information structure for a concept (see pattern_3 in Fig. 5; it is used to recognize the pages for car rental). In the simplest case, a pattern can be a key word and in this case an EO is degenerated to an ontology. Such a pattern is used to find pages relevant to a concept from the internet.

To find the remote information sources related to a concept, we need a mechanism to recognize web pages. Normally, one can determine the similarity of two pages in different ways. For instance, one can use the information retrieval notion of textual similarity [Sal83]. One could also use data mining techniques to cluster pages into groups that share meaningful (e.g., [PE98]), and then define pairs of pages within a cluster to be similar. A third option is to compute textual overlap by counting the number of chunks of text (e.g., sentences or paragraphs) that pages share in common [SGM95, SGM96, BGM97, BB99, CSG99]. In all these schemas, there is a threshold parameter that indicates how close pages must be to be considered similar (e.g., according to number of shared words, *n*-dimensional distance, number of overlapping chunks). This parameter needs to be empirically adjusted according to the target application.

Web recognizer

All the methods mentioned above don't, however, pay attention to an important aspect of information: the structure of a page. As we know, a page in HTML or XML format always consists of a hierarchical structure, starting with a root element as shown in Fig. 3(a).

Such structure information can be used to speed up page matchings (since taking the structure of pages into account can limit the search for similar terms to small parts of a text). A frequently used technique to explore the similarity of structures is *tree matching*; but it is too strict and a similar page may be filtered out undesirably. So we introduce a more relaxed concept: *tree inclusion*, which can be defined as follows.

Definition 1 (*labeled tree*) A tree is called a labeled tree if a function *label* from the nodes of the tree to some alphabet is given, or say each node in the tree is labeled.

Definition 2 (*tree inclusion*) Let T_1 and T_2 be two labeled trees. A mapping M from the nodes of T_2 to the nodes of T_1 is an embedding of T_2 into T_1 if it preserves labels and ancestorship. That is, for all nodes u and v of T_2, we require that

a) $M(u) = M(v)$ if and only if $u = v$,
b) $label(u) = label(M(u))$, and
c) u is an ancestor of v in T_2 if and only if $M(u)$ is an ancestor of $M(v)$ in T_1.

Fig. 6. Sample XML pages

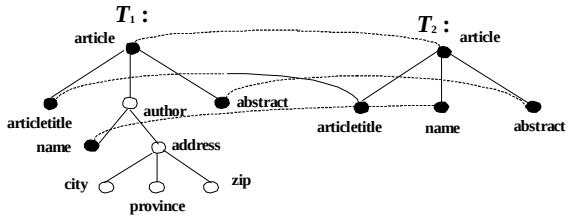
```
<article>
   <articletitle>On the DTD Mapping into OO Schemas</aricletitle>
   <author id="dawkins">
            <name>
                <firstname>Richard</firstname>
                <lastname>Dawkins</lastname>
            </name>
            <address>
                <city>Winnipeg</city>
                <province>Manitoba</province>
                <zip>R3B-2E9</zip>
            </address>
   </author>
   <abstract>... database ... SGML ... informatics ... </abstract>
</article>

<article>
   <articletitle>DTD and OO Schemas</aricletitle>
   <author id="dawkins">
            <name>
                <firstname>Richard</firstname>
                <lastname>Dawkins</lastname>
            </name>
   </author>
   <abstract>... DB ... XML ... computer science ... </abstract>
</article>
```

Here, the mapping M can be implemented as a method discussed in [Sal83] or any method used in [SGM95, SGM96, BGM97, BB99, CSG99]. For example, the XML pages shown in Fig. 6(a) and 6(b) can be represented as two trees shown in the left and right parts of Fig. 7(a), respectively. If a mapping as shown in Fig. 7(b) can be determined, we know that the tree shown in the right part of Fig. 7(a) is included in the tree shown in the left part of Fig. 7(a) (see the dashed lines in the figure). In this case, we claim that the page shown in Fig. 6(b) is similar to the page shown in Fig. 6(a).

Note that to get an equation such as '$M(T_1$.articletitle) = T_2.articletitle', we must check the similarity between 'On the DTD Mapping into OO Schemas' and 'DTD and OO Schemas'. Similarly, to get '$M(T_1$.abstract) = T_2.abstract', we have to check the similarity between 'database' and 'DB', 'SGML' and 'XML', and 'informatics' and 'computer science'. We can utilize the technique discussed in [Sal83] to determine the similarity of T_1's abstract and T_2's abstract based on the common words in these two paragraphs.

We note that a node in a tree can be a complex structure. In fact, it can be a subtree by itself. Therefore, to decide whether two nodes are matching, we can check their 'tree inclusion', leading to a concept of the so-called *recursive tree inclusion*, by which the tree inclusion is utilized recursively. Another important concept is the *inclusion degree*, which is used to measure to what extent a tree includes another one.



(a)

$M(T_1$.article) = T_2.article
$M(T_1$.articletitle) = T_2.articletitle
$M(T_1$.name) = T_2.name
$M(T_1$.abstract) = T_2.abstract

(b)

Fig. 7. Illustration for tree inclusion

This is needed to apply the recursive tree inclusion to test similarities. These two concepts will be discussed in great detail in a paper in preparation.

CONCLUSION

In this paper, we have discussed the system architecture of a WDDB, which is composed of three parts: a local document database, a web connector, and a web recognizer. The local document database can be considered as an information source reachable through the network. It can also connect to some other document databases through its web connector, which maintains a set of URLs. Each URL is related to a concept or a pattern that specifies the content of the demote database. The task of the web recognizer is to perform web recognition. It works as a web wrapper [AMM97, Hs98] but is more powerful in the sense that it recognizes a web page by checking not only part of the page's syntactic structure but the whole page with semantics considered. It will associate a set of URLs with a concept or a pattern which indicates the contents of the document databases pointed to by the URLs.

REFERENCES

Ab97 S. Abiteboul, Querying semi-structured data, in *Proc. Int'l Conference on Data Engineering (ICDE)*, 1997. http://www-db.stanford.edu/pub/papers/icdt97.semistructured.ps.

ACC97 S. Abiteboul, S. Cluet, V. Christophides, T. Milo, G. Moerkotte and J. Simeon, "Querying documents in object databases," *Int. J. on Digital Libraries*, Vol. 1, No. 1, Jan. 1997, pp. 5-19.

AMM97 P. Atzeni, G. Mecca and P. Merialdo: Semistructured and structured data in the web: going back and forth, *Proc. of ACM SIGMOD Workshop on Management of Semi- structured Data* (1997), pp. 1-9.

BGM97 A.Z. Broder, S.C. Glassman and M.S. Manasse, Syntactic clustering of the web, in *Proc. of 6th Int. World Wide Web Conference*, April 1997, pp. 391-404.

CA98 Y. Chen, K. Aberer, Layered Index Structures in Document Database Systems, *Proc.* 7th *Int. Conference on Information and Knowledge Management (CIKM)*, Bethesda, MD, USA: ACM, 1998, pp. 406-413.

CA99a Y. Chen and K. Aberer, Combining Pat-Trees and Signature Files for Query Evaluation in Document Databases, in: Proc. of *10th Int. DEXA Conf. on Database and Expert Systems Application*, Florence, Italy: Springer Verlag, Sept. 1999. pp. 473-484.

CA99b Y. Chen and K. Aberer, SGML DataBlade - A Document Database System, in: *Proc. of Int. Symposium on Database Application in Non-Traditional Environments*, Tokyo, Japan, IEEE, Dec. 1999, pp. 37-40.

Ch02 Y. Chen, A New Way to Speed-up Recursion in Relational Databases, in: *Proc. of 13th Information Resources Management Association Intl. Conference*, Seattle, USA, May 19-22, 2002, pp. 356-360.

CCS00 V. Christophides, S. Cluet and J. Simeon, "On Wrapping Query Languages and Efficient XML Integration," in *Proc. of the ACM SIGMOD Conf. on Management of Data*, pp. 141-152, 2000.

Co99 Copernic: http://www.copernic.com.

CSG99 Cho, N. Shivakumar, H. Garcia-Molina, "Finding replicated web collections," http://dbpubs.stafford.edu/pub/1999-64.

DD94 S.J. DeRose and D.D. Durand, "*Making Hypermedia Work: A User's Guide to HyTime*," Kluwer Academic Publishers, London, 1994.

Gr94 I.S. Graham: HTML-documentation and style guide, http://www.utirc.utoronto.ca/ HTMLdocs/NewHTML/htmlindex.html, 1994.

Hs98 C. Hsu: Initial results on wrapping semistructured web pages with finite-state transducers and contextual rules, *Proc. of AAAI-98 Workshop on AI and Information Integration* (1998), pp. 66-73.

Kn73 D.E. Knuth, The Art of Computer Programming: Sorting and Searching, Addison-Wesley Pub. London, 1973.

Kn99 KnowAll: http://www.worldfree.net.

KS95 D. Konopnicki and O. Shmueli, W3QS: A query system for the world-wide web, in *Proc. of the 21st VLDB Conference*, Zurich, Switzerland, 1995, pp. 54-65.

Le94 T.B. Lee: RFC 1738: Uniform Resource Locators, http:/

/www.w3.org/hypertext/ WWW/Addressing/rfc1738.txt, Dec. 1994.

PE98 M. Perkowitz and O. Etzioni, Adaptive web sites: automatically synthesizing web pages, in *Proc. of 15th National Conf. on Computer and Human Interaction (CHI'97)*, 1997.

Sal83 G. Salton, *Introduction to modern information retrieval*, McGraw-Hill, New York, 1983.

SGM95 N. Shivalumar and H. Garcia-Molina, SCAM: a copy detection mechanism for digital documents, in *Proc. of 2nd Int. Conf. on Theory and Practice of Digital Libraries (DL'95)*, Austin, Texas, June 1995.

SGM96 N. Shivalumar and H. Garcia-Molina, Building a scalable and accurate copy detection mechanism, in *Proc. of 1st Int. Conf. on Digital Libraries (DL'96)*, Bethesda, Maryland, March 1996.

Sq99 Squid: http://www.squid-cache.org.

TM01 T. Fiebig and G. Moerkotte, "Algebraic XML Construction in Natix," in *Proc. of the 2nd Int. Conf. on Web Information Systems Engineering*, pp. 250-259, 2001.

VAB96 M. Volz, K. Aberer and K. Böhm, "Applying a Flexible OODBMS-IRS_Coupling to Structured Document Handling," *Proc. of 12th Int. Conf. on Data Engineering (ICDE)*, New Orleans, 1996, pp. 10-19.

W3C98a World Wide Web Consortium, Extensible Markup Language (XML) 1.0. http// www.w3.org/TR/1998/REC-xml/19980210, Febuary 1998.

W3C98b World Wide Web Consortium, Extensible Style Language (XML) Working Draft, Dec. 1998. http//www.w3.org/TR/1998/WD-xsl-19981216.

Wi92 G. Wiederhold, "Mediators in the Architecture of Future Information Systems," *IEEE Computer*, 25:38-49, 1992.

WL00 K. Wang and H. Liu, Discovering structural association of semistructured data, *IEEE transaction on knowledge and data engineering*, Vol. 12, No. 3, May/June 2000, pp. 353-371.

query evaluation