http://ansinet.com/itj

ITJ

ISSN 1812-5638

INFORMATION TECHNOLOGY JOURNAL

ANSInet
Asian Network for Scientific Information
308 Lasani Town, Sargodha Road, Faisalabad - Pakistan

Information Technology Journal 4 (1): 21-31, 2005
ISSN 1812-5638

Design and Implementation of a Semantic Document Management System

Guoren Wang, Bin Wang, Donghong Han and Baiyou Qiao
College of Information Science and Engineering, Northeastern University,
Shenyang 110004, People's Republic China

Abstract: Easily accessible information on the World Wide Web (WWW) and affordable large capacity secondary storage make it easy to build up very large document collections even in personal computers. However, the method of organizing files in computers has not been changed too much for decades. Searching for a particular document or file from a gegabytes collection based on traditional tree structured file directories becomes never an easy task. This study presents a system where documents are no longer identified by their file names. Instead, a document is represented by its semantics in terms of descriptor and contents vector. The descriptor of a document consists of a set of attributes, such as date of creation, its type, its size, annotations, etc. The content vector of a document consists of a set of terms extracted from the document. Such semantic information provides the user with associative searching capability, that is, documents can be obtained by giving required properties. The representation of document semantics and document organization and key word-based indexing techniques are discussed. Furthermore, for the largely used XML data in Web representing and exchanging, some structure-based querying techniques are proposed in this study, i.e. structural indexes and path expression optimization principles. A prototype visual based explorer that makes use of semantics of documents is also described.

Key words: Document databases, semantic documents, document clustering, path expression querying, visual document exploring

INTRODUCTION

With the rapid development of the Internet and the World Wide Web (WWW)[1,2], very large amount of information is available and ready for downloading, most of which are free of charge. At the same time, secondary storage device, the hard disk with large capacity is available at affordable prices. Disk space seems no longer a concerns even for personal computer users. Most of us nowadays often dump a large number of various types of documents into our computers without much thinking. On the other hand, the file system has not been changed too much for the past decades. Although there are various types of file systems most of them organize files in directories that form a tree structure and a file is identified by its name and pathname in the directory tree. Finding a particular file on the disk with dozens of gegabytes data becomes never an easy task.

The problem has attracted attentions from both the academic researchers and indus-trial vendors. One of possible solutions is to introduce semantics of files into file systems[3]. By semantics of files, it is meant any higher-level information related to the file. Vasudevan and Pazandak listed the following example of such information[4]:

- Definitional: e.g. file extension and magic numbers define file type;
- Associative: e.g. keywords in file that characterize content;
- Structural: e.g. physical and logical organization of the data, including intra-and inter-file relationships; Behavioral: e.g. viewing and modification semantics, change management and
- Environmental: e.g. creator, revision history.

With such built-in information, tools that use such information can be built to provide users with more flexible way to access files other than files names and paths in directories.

In the traditional file systems, the content-based/keyword-based querying is widely used to make users searching their interests with some keywords in the documents. While there are a growing amount of Web documents that are semi-structured documents, especially XML documents. The XML data is proposed as the new

Corresponding Author: Guoren Wang, College of Information Science and Engineering, Northeastern University, Shenyang 110004, People's Republic China E-mail: wanggr@mail.neu.edu.cn

standard of data representing and exchanging in the Web. For this kind of data, the conventional content-based querying is not enough as there could be some queries like "find all peoples who live in New York and buy more than 10 books per-year". These queries need to check the structure of documents and perform the querying. We call the procedure of processing document structure related queries as structure-based querying. The various XML query languages proposed for XML data are all structure-based[5-7]. The path expressions are adapted as one of the core components of them to reach the arbitrary depth of the data. And some algorithms and index structures are already proposed to support efficient structure-based querying on semi-structured data[8-11].

The present study is along the same line: managing large document collections using semantics. Among those semantic information mentioned above, two types of information, descriptive and associative ones were used. The rationale is that when a user searches for a document stored in her/his computer, s/he most likely has some vague idea about the document. The typical information includes the source of the document (creators, places where the document was obtained), its type (text, html, word, postscript, etc.), the topics and its main theme, etc., though most of time such information s/he can remember is not precise, while the most difficult thing to remember is the name of the file that contains the document. On the other hand, in order to support the structure-based querying on semi-structured data, i.e. XML data, we adopted Xpath[5] as the query language and some index structures and query optimization techniques are proposed to efficiently evaluate XPath queries. A prototype system has been developed to demonstrate the feasibility and merits of such a system. Comparing to existing systems and tools, our system has the following unique features.

- File name becomes optional and a document in the system is no longer identified by the name of the stored file. Instead, semantic information, such as document descriptor consisting of a number of attributes and a content vector consisting of a set of terms contained in the document are used to capture the semantics of a stored document. Flexible associative query about the document based on such semantics becomes possible.
- In order to support semantic based exploring and retrieving of stored documents efficiently, R*-tree index is built on scalar descriptive attributes and inverted index is built for string attributes of the descriptor and content vector. Furthermore, not only text-based documents, but also some non-text document, such as postscript and pdf files, are also full-text indexed.
- Documents are clustered into topics based on semantics so that only portions of the collection will be searched for a particular query.
- Structural indexes are built on XML documents to support structure-based queries with XPath language. Meanwhile, two path expression optimization techniques, named path-shorten and path-complementing, are applied.
- A document is viewed as a point in the multidimensional space defined by the attributes in its descriptor. A visual document exploring tool, X-Explorer, allows users to browse the documents over the multidimensional document space.

The preliminary experiments indicated that the system achieves its goal and its performance is satisfactory.

MANAGING DOCUMENT COLLECTION USING SEMANTICS

This study defines what kinds of semantics should be supported by the semantic document system and then discuss how to extract the semantics from documents, finally discuss how to index the extracted semantics.

Semantics of documents: General speaking, semantics of a document include any information regarding the document and its relationship with other documents. Among various kinds of semantic information about a document, we are in particular interested in two types of semantics, definitional and associative information because (1) they can help to retrieve documents more efficiently and (2) they are easy to remember by users, though may not be precise.

Definitional semantics of a document refers to descriptive information about the document. We use a Document Descriptor (DD) to represent such information. In our system, the document descriptor has the following structure:

```
enum FileType {PS, PDF, XML, TXT};
typedef struct {
d_Ref <Document> XMLId; // for XML documents
d_Binary fileId;              // for other files
}DOCID;
class descriptor: public d_Object{
private:
d_String createdTime;     // time when the document is
                          created
d_ULong fileSize;         // size in bytes
FileType fileKind;        // type of document
d_UShort clusterNo;       // the cluster to which the doc
                          belongs
d_String title;           // title of the document
```

```
d_String source;        // source of the file
d_String memo;          // annotation to the document
DOCID docId;            // the ID of document public:
...
};
```

It can be seen from the structure that the document descriptor contains meta-data about the document. Most attributes in the descriptor are self-explainable. Attribute fileKind specifies the type of stored document. What listed are the types of documents implemented in the current prototype system. It is obvious that the list can grow to include more types of documents. The use of attribute clusterNo will be described in later sections. Attributes title, source and memo are string attributes containing user provided information about the document. The values of those attributes in the descriptor can be initialized when the document is stored in the database. For example, title and source can be induced from the link of documents when they are downloaded from the Web. The contents of memo can be obtained from the user who downloads/creates the document. Note that there is no logical name for a document because a document is stored in a database as an object rather than a file in the file system. Another reason of not including name as one attribute is that unless strict naming rule is enforced, a user tends to forget the name of a document created earlier.

Associative information: Attributes title, source and memo in the document descriptor provide certain associative information that helps the retrieval of documents based on their contents. However, such information is limited. A user may remember the contents of a previously seen or downloaded document vaguely. The keywords remembered may not be in the attributes. The probability of those words appeared in the document is much higher. Such information for associative retrieval of a document is captured by a vector of terms appeared in the document, denoted by Content Vector (CV).

Structural information: Some documents are structured or semi-structured, such as XML documents. It is possible to search for documents based on such structural information. We use the schema information/DTD as the schema constraints of XML documents and guide the structure-based querying. In this paper, we presented how to query the XML data based on their structure information with XPath language. It is our belief that with those information, stored documents can be accessed through various querying facilities, including query languages, keyword based search and explorative browsing.

System reference architecture: We first briefly describe the reference architecture of a system where documents are managed based on their semantics, followed by extraction of semantic information from downloaded documents.

The system, as shown in Fig. 1, is based on the client/server paradigm. The server side consists of an X-Server, the document databases, Metadata and Indexes. We use prefix X- because XML is used as a common language to specify the documents and metadata where exchange of data is involved. Documents are stored in a document database system implemented on the top of an object database system[12-14] conforming to the ODMG Standard[15]. Definitional semantics, document descriptor DD and structural semantics, document schema DS, are stored as metadata. Associative semantics, the Content Vectors (CV) are in fact a logical concept. They are stored in the form of inverted index to provide associative retrieval. The X-server consists of modules for storing a

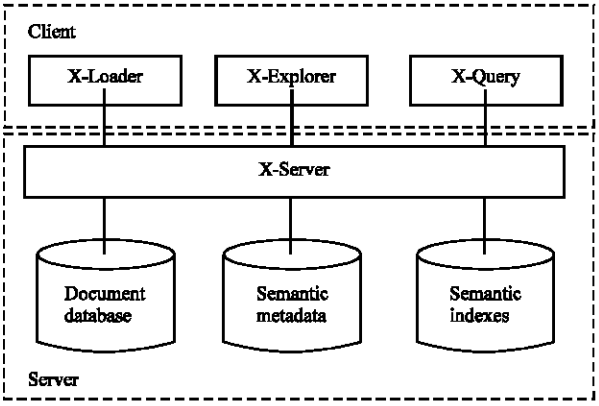

Fig. 1: Managing document collections using semantics

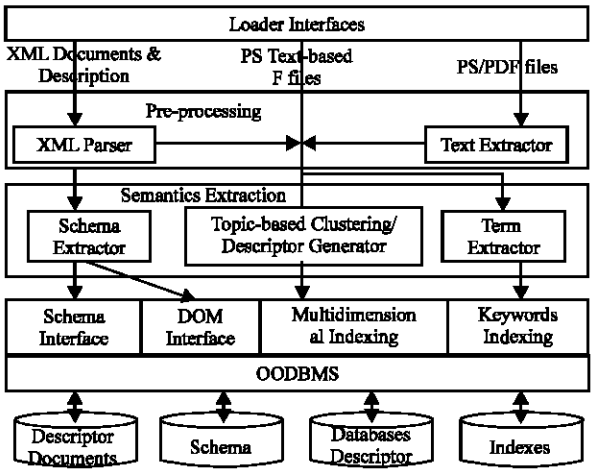

Fig. 2: Storing documents and extracting semantic information

document into database, inserting semantic information into metadata and index, processing user queries and interfacing with the client side.

The client side consists of X-loader and querying facilities, X-query and X-explorer. X-loader is responsible for loading documents and extracting their semantics. X-query accepts queries expressed in XML-based query languages, i.e. X-path and X-explorer is a multidimensional document explorer to help users browse and locate documents they want. The content-based/keyword-based queries are integrated into X-explorer by regarding the keyword as part of X-explorer conditions.

Storing documents and extracting their semantics: Figure 2 depicts the process of extracting semantic information and storing it with documents in the system. All documents and related information are managed by an OODBMS.

The loader interface interacts with the user and Internet Browser which is the default browser of the system. When a user drags a link pointing to a document of her/his interests from the Internet Explorer to the X-Loader, the anchor text (treated as the title) and URL of the link are captured using the Microsoft OLE drag-drop technology. The document is then downloaded from the source site. There are two major steps after the documents are downloaded and before they are stored in the database.

The first step preprocesses a document. In case that the document is compressed, it will be decompressed first for semantic information extraction. The type of the document is recognized (analyzed), currently based on the file extensions. Documents can be categorized into three groups: XML documents (semi-structured documents), text-based documents and non-text based documents such as postscript and pdf files. For semi-structured documents, schema information is extracted. For non-text based documents, text extraction is applied.

The second step, semantics extraction, generates semantic information for the document. Most scalar attribute values, such as creation time, size and etc., are easy to be obtained. If necessary, the value of clusterNo, is obtained after the document is clustered into topics using the method described in the next section. The document descriptor is generated. In order to obtain the associative semantics of documents, the text representation of a document is gone through the traditional term extraction process. Those terms form the context vector of the document. This term extraction process consists of typical procedures such as removing stop words, stemming, term selection, etc. For semi-structured documents, such as XML documents, the schema information is extracted and the structural indexes are built to support structure-based queries.

Both the document and related information are stored in a database managed by an OODBMS. To support efficient access to the stored documents, the scalar attributes in the document descriptor are indexed using a multidimensional index and the string attributes in the descriptor and content vector are indexed by a keyword index, implemented as an inverted index.

SUPPORTING EFFICIENT RETRIEVAL OF THE STORED DOCUMENTS

Two techniques, clustering and indexing, are used to support efficient retrieval to the large volume of documents stored in the database.

Topic based document clustering: Document clustering is a well-known technique used in information retrieval. The problem was re-examined in the recent surge of Web[16] data mining research[17,18]. In our context, document clustering can be used to improve the retrieval performance in two ways. First, with the hypothesis that similar documents are often accessed together, documents in the same cluster are physically clustered in the database so that disk seeking time is expected to be reduced. Second, clustering information can be used by users so that the search space of required document can be effectively reduced. With our objectives in our mind and inspired by Schutze and Silverstein's projection approach[19], we developed a topic-based clustering approach, which is proved an effective clustering method for document clustering. It works as follows.

- Different from traditional clustering methods where no pre-defined cluster definitions are available, topic-based clustering has a set of topic phases that are used for clustering. This set of topics can be defined by either users or system designers. The rationale is as follows. The pre-determined topic phrases can guarantee that clustering would be done according to users' intention and the features of clusters are interesting to users.
- Based on the topic phrases users specified, a document is represented a n-dimensional vector x in the following formula (1), where n is the number of the topic phrases, t_i is the i^{th} topic, F_{ti} is the frequency of t_i appearing in the document (if topic t_i appears in the title of the document, then F_{ti} is weighted, because the title probably reflects the feature of document), xi is the i th component of vector x for a document.

$$x = < x_1, x_2, ..., x_n > \text{ where, } x_i = \sqrt{F_{ti}} \quad (1)$$

- The document space is divided into k+1 clusters. The documents with $F_{ti} = 0 (1 \le i \le n)$ are grouped into the same cluster called the miscellaneous cluster and the left documents are divided into k clusters that optimize the following criterion function (2).

$$E = \sum_{i=1}^{k} \sum_{x \in C_i} d\ (x, m_i) \qquad (2)$$

In the above equation, m_i is the centroid of cluster C_i while $d(x, m_i)$ is the euclidean distance between x and m_i. Thus, intuitively, the criterion function E attempts to minimize the distance of every point from the mean of the cluster to which the point belongs.

First, 2n+1 initial cluster centers are determined by the following formula (3), where $\bar{x}_i$ is the average of the i^{th} component of all document vectors, s_i is the standard deviation of the i^{th} component of all document vectors.

$$\begin{cases} m_0 = [\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n] \\ m_1, m_2 = [\bar{x}_1 \pm \sigma_1, \bar{x}_2, \ldots, \bar{x}_n] \\ \ldots \\ m_{2i-1}, m_{2i} = [\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_{i-1}, \bar{x}_i \pm \sigma_i, \bar{x}_{i+1}, \ldots, \bar{x}_n] \\ \ldots \\ m_{2n-1}, m_{2n} = [\bar{x}_1, \bar{x}_2, \ldots, \bar{x}_n \pm \sigma_n] \end{cases} \qquad (3)$$

Then, the nearest pair of clusters is merged until the desired number of cluster is achieved.

Finally, we use the iterative algorithm of k-means[20] to compute the final cluster centers to minimize formula E.

- Summarize the features of clusters excluding the miscellaneous cluster. For a given cluster, the m ($1 \le m \le n$) topic phrases with maximal average frequency are selected to characterize the cluster.

Because the number of dimensionalities representing documents is reduced, it is not doubted that the response time of the clustering algorithm is dramatically improved because the cost of distance computation is improved by reducing the number of dimensions. In addition, the topics based clustering approach overcomes the disadvantage of losing valuable information of word-based approaches.

To evaluate the effectiveness of the algorithm and the quality of the clustering obtained, we presented here one sample experiment. We use research papers in the proceedings of SIGMOD'97 as testing data. Based on the sessions appeared in the proceedings, 19 topics are selected to model the documents. The selected topics, the

Table 1: Selected topics and original grouping situation of sessions

Session no.	Selected topics	No. of papers	Paper encoding
S01	Similarity	3	1-1, 1-2, 1-3
S02	Index	6	2-1, 2-2, 2-3, 2-4, 2-5, 2-6
S03	Data cube, OLAP	3	3-1, 3-2, 3-3
S04	Performance, Benchmark	3	4-1, 4-2, 4-3
S05	Aggregate, Derived data	3	5-1, 5-2, 5-3
S06	Distributed data	3	6-1, 6-2, 6-3
S07	Query, Sort	3	7-1, 7-2, 7-3
S08	Data ,mining	3	8-1, 8-2, 8-3
S09	Data access	3	9-1, 9-2, 9-3
S10	Spatial Data	3	10-1, 10-2, 10-3
S11	View Maintenance	3	11-1, 11-2, 11-3
S12	Rule	3	12-1, 12-2, 12-3
S13	Security, Commit, Transaction	3	13-1, 13-2, 13-3

Table 2: Clustering result and cluster features

Cluster no.	Cluster features	Papers
C01	Similarity, Query	1-1, 1-2
C02	Index, Performance	2-1, 2-2, 2-3, 2-4 , 2-5, 2-6
C03	Rule, Performance	1-3, 9-3
C04	Data cube, Query	3-1, 3-2, 3-3
C05	Benchmark, Performance	4-1, 4-2, 4-3
C06	Derived data, Aggregate	5-1, 5-2, 5-3
C07	Distributed data, Query	6-1, 6-2, 6-3
C08	Query, Sort	7-1,7-2, 7-3, 9-1, 9-2
C09	Data mining, Rule	8-1, 8-2, 8-3
C10	Spatial data, Query	10-1, 10-2, 10-3
C11	View maintenance, transaction	11-1, 11-2,11-3
C12	Rule, transaction	12-1, 12-2, 12-3
C13	Transaction, Security	13-1, 13-2, 13-3

Table 3: Confusion matrix

Proceedings sessions	Clusters gotten by the algorithms												
	C01	C02	C03	C04	C05	C06	C07	C08	C09	C10	C11	C12	C13
S01	2	0	1	0	0	0	0	0	0	0	0	0	0
S02	0	6	0	0	0	0	0	0	0	0	0	0	0
S03	0	0	0	3	0	0	0	0	0	0	0	0	0
S04	0	0	0	0	3	0	0	0	0	0	0	0	0
S05	0	0	0	0	0	3	0	0	0	0	0	0	0
S06	0	0	0	0	0	0	3	0	0	0	0	0	0
S07	0	0	0	0	0	0	0	3	0	0	0	0	0
S08	0	0	0	0	0	0	0	0	3	0	0	0	0
S09	0	0	1	0	0	0	0	2	0	0	0	0	0
S10	0	0	0	0	0	0	0	0	0	3	0	0	0
S11	0	0	0	0	0	0	0	0	0	0	3	0	0
S12	0	0	0	0	0	0	0	0	0	0	0	3	0
S13	0	0	0	0	0	0	0	0	0	0	0	0	3

original grouping of sessions are shown as in Table 1. Note that two sessions about indexing in the proceedings are merged, so there are totally 13 sessions with 6 papers in session 2.

Table 2 gives the clustering result and the feature of each cluster. For each cluster in Table 2 except for C03 there exist a corresponding session. For example, C02, C04, C06 correspond to S02, S03, S05, respectively and the cluster and corresponding session have similar features. Cluster C03 should correspond to session S09 about data access. Because topic query appears in papers 9-1 and 9-2 many times, the two papers are grouped into cluster C08. As for papers 1-3 and 9-3, topics except for rule and performance do not almost appear in the two papers and therefore they are grouped into the same cluster and the features of the cluster are rule and performance. Actually, the frequencies of all topics in the two papers are very low, so the two papers cannot be characterized by the topics appeared in the titles of sessions in the proceedings.

Table 3 gives the confusion matrix for the clustering result and the pre-defined groupings. We can see that all papers except for papers 1-3 and 9-3 are grouped into clusters correctly and therefore the quality of clustering decreases by about 5%. The main reason of incorrectly grouping the two papers is that the selected topics did not capture the features of these two papers.

Indexing the semantic information: Two types of semantic information, the Document Descriptor (DD) and Content Vector (CV) are indexed to support fact access. Based on the properties of the values of DD and CV, two indexing structures are supported: Four scalar attributes, createdTime, fileSize, fileKind and clusterNo, are indexed using R*-tree[21]. R*-tree is an efficient multidimensional indexing method when number of dimensions is not too high[3-5]. Terms in string attributes in DD, title, source and memo, together with the content vector, are indexed using a B+-tree. The leave nodes of the tree contain entries of terms associated with a list of ID's of documents in which the term appears.

STRUCTURE-BASED QUERYING XML DOCUMENTS

As the popularity of using XML data as data representing and exchanging standard on the WWW, we take XML documents as the main target of the structure-based query processor X-Query.

X-Query interface: Most query languages for XML data proposed recently use path expressions to catch the tree structure of XML data for path expression can reach the arbitrary depth of XML data tree. In XBase, we used XPath as the query language, which fully defined the semantics of querying XML data with path expressions. Figure 3 is the screen dump of the interface of X-Query.

The interface of X-Query is shown in Fig. 3, which mainly consists of four parts: Document Selection Window (DSW), DTD Displaying Window (DDW), Path Expression Construction Window (PECW) and Result Displaying Window (RDW). Users first can select a document they want to query from the document selection window via the left-up selection window.

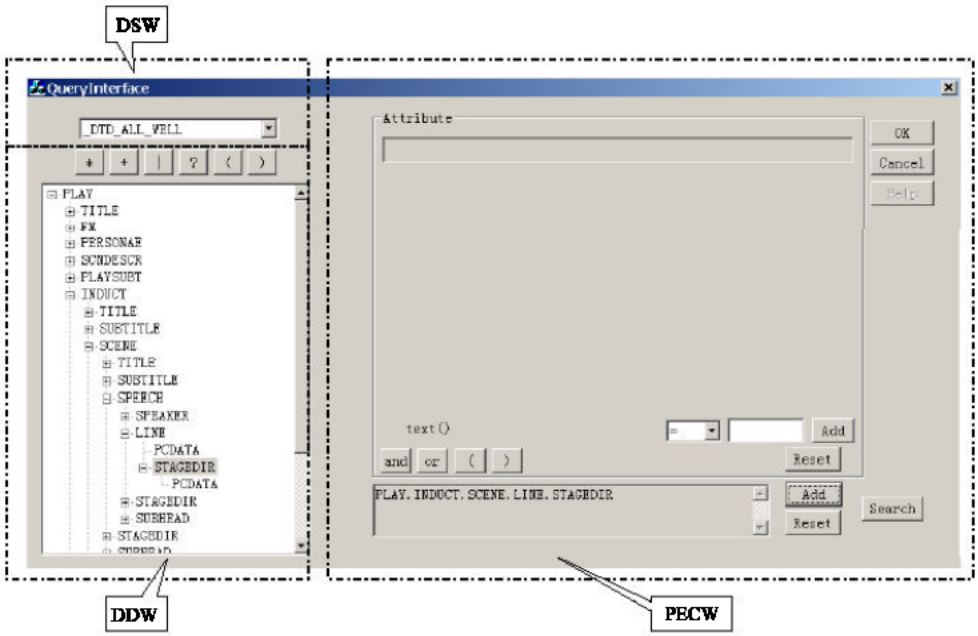

Fig. 3: The X-Query interface

Once the document is selected, the DTD corresponding to the selected document is shown in the DTD displaying window, based on which users can construct an RPE-based query through the path expression construction window. The path steps are determined by the click operation in the DDW and are shown at the bottom of PECW. The attributes of the current selected path step are listed on the top part of PECW and users can specify predicated on attributes and element texts in this window. After the query is constructed, it can be executed by clicking the search button on the right-bottom and the result is constructed as a XML document that is shown in RDW, a pop-up window.

Data model and data type model: The Document Object Model (DOM) is an Application Programming Interface (API) for XML and HTML documents, which defines the logical structure of documents and the way a document is accessed and manipulated. In DOM, the data are abstracted into entities (elements and attributes) and these entities are organized together via the parent-children and element-attribute relationship to form a data tree, i.e. DOM tree. Formally, we modeled DOM as a node-labeled tree $T_d = (V_d, E_d, \delta_d, \sum_d, root_d, oid)$, where V_d is the node set including element nodes and attribute nodes; E_d is the set of tree edges that denote the parent-children relationship between elements and the element-attribute relationship between elements and attributes; δ_d is the mapping function from nodes to nodes that actually are the relationship constraints. Every node has a unique name that is a string-literal of $\sum_d$ and a unique identifier that comes from identifier set oid. Finally, every XML data tree has a root element rootd that is included in V_d.

Though XML data is self-descriptive, the Document Type Definition (DTD) is proposed by W3C to further and explicitly constraint the relationship between XML elements, e.g. an element should contain what kind of or/and how many sub-elements. It mainly defines the parent-children relationship between XML elements and the order between the sub-elements of an element. In this study, we model XML DTD as a directed, node-labeled graph, in which the order constraint is not considered. We will explain the reason that the order constraint is omitted later in the following sections. Formally, a DTD is defined as a directed graph $G_t = (V_t, E_t, \delta_t, \sum_t, root_t)$. V_d is the node set including element type nodes. E_d is the set of graph edges that denote the element-subelement relationship between elements. Attributes are parts of elements. δ_d is the mapping function from nodes to nodes that actually constraint which element can contain which sub-elements. Every node has a unique name that is a string-literal of $\sum_d$ and this name is actually element type name. Finally, every XML DTD graph has a root element $root_t$ that is included in V_t, which is defined as the node with only outgoing edges and without any incoming edges.

Extent join algorithm and structural indexes: There are many approaches proposed to evaluate path expression queries. Among them the extent join algorithm is one of the efficient ways. In extent join, path expression queries first are decomposed into several path steps, then for each path step a temporary result set is computed and finally a multi-way join between these temporary result sets are performed to evaluate the whole query. Conceptually, the temporary result sets of path steps are element extents that are defined below. All the definitions are based on an XML data tree $T_d = (V_d, E_d \delta_d, \sum_d, root_d, oid)$ and a corresponding DTD graph $G_t = (V_t, E_t, \delta_t, \sum_t, root_t)$.

We should see some useful notations before define XML element extent. First, pc-pair(pid, cid) is a pair of element identities in which pid, cid $\in$ oid and pid is the parent of cid and adpair(aid,did) is a pair of element identities, in which aid, did $\in$ oid and aid is the ancestor of did. A pcpair is a specialized case of an adpair. Additionally, element is defined to act as any element. Then we have the XML element extent and the path constrained element extent.

Definition 1: XML element extent (Ext). All pcpairs of a given tag name Tag is represented by Ext(any, Tag) = {pcpair(pid, id)\id is an instance of Tag ^ pcpair(pid, id) is true} while all adpairs of two given tag names an and dn can be represented by Ext(an,dn) = {adpair(aid, did) \ pcpair(ε,aid) $\in$ Ext(any, an) ^ pcpair(ε,did) $\in$ Ext(any, dn) ^ adpair(aid, did) is true}.

Definition 2: Path constrained element extent (PCExt). PCExt(an; dn; P) = {adpair(aid, did) \ adpair(aid, did) $\in$ Ext(an, dn) ^ did $\in$ P(aid)}, where P(aid) is the element instance set that can be reached from aid via path P.

The basic idea of extent join algorithm is replacing the tree traversal procedures by the set join operations. Before the whole path expression is evaluated, the intermediate result sets to be joined must be first computed that are actually Exts and PCExts. And then the ancestor-descendant/parent-children relationship based multi-join operation is performed to evaluate the whole query. For the XML extents consist XML ancestor-descendant/parent-children element pairs, so the joins need not to check the relationships between sets to be joined and therefore are very straightforward.

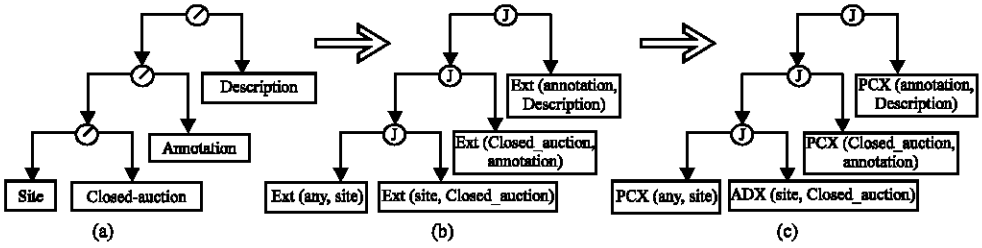

Fig. 4: Path expression (a) extent join tree (b) and execution tree (c)

To support the extent join algorithm, structural indexes are required to maintain the parent-children and ancestor-descendant relationship with the index results. XBase system totally implemented three structural indexes: Ancestor-Descendant Index (ADX) and Parent-Children Index (PCX). ADX is used to index Ext (Pname, Cname) for given element names Pname and Cname where Pname is the ancestor of Cname. ADX actually indexes the ancestor-descendant relationship between elements with specified element types. PCX is used to index PCExt (Pname, Cname, "Pname/Cname") for given element names Pname and Cname where Pname must be the parent of Cname, which preserves the parent-children relationships.

About the indexes above, only the principles are introduced. The implementations of them are relatively simple for they have no special demands on the index structures. The traditional index structures, e.g. B+ tree, are fit for these indexes.

Consider path expression query "/site//closed auction/annotation/description" containing three path steps and two connectors. As Fig. 4 shows, each path step corresponds to an intermediate results set, i.e. an element extent; each connector is transformed into a join operation and the results of joins are the path constrained element extents. For example, the join between Ext (any, site) and Ext (site, closed-auction) is PCExt (site, closed auction, "/site/closed auctions/closedc auction") and the PCExt acts as a intermediate result used to perform another join with Ext (closed auction, annotation) to get yet another PCExt. Path expression queries must be transformed into evaluation plans to get evaluated. And the art of transformation is focused on the path steps to corresponding extents and the following shows the full transformation rules.

- Connectors ("//" and "/") are transformed into joins between two sets.
- Path step "*" is rewritten with element types using the mapping function δ_t; e.g. δ_t(site) = {regions, people, closed auctions, open auctions} and path expression "/site/*/person" should be rewritten as "/site/(regions\people\closed-auctions\open-auctions)/person".
- Path steps with connector "//" ahead are transformed into a corresponding ADX operator; e.g. for path step S_2 in "S_1//S_2" is transformed into ADX(S_1, S_2).
- Path steps with connector "/" ahead are transformed into a corresponding PCX operator; e.g. for path step S_2 in "S_1/S_2" is transformed into PCX(S_1, S_2).
- Path steps containing " \ " are transformed into the unions of corresponding indexes; e.g. path step (S_2\$S_3$) of "$S_1$/($S_2$\$S_3$)" is transformed into PCX(S_1, S_2) $\cup$ PCX(S_1,S_3).

Optimizing path expression queries: We have introduced the basic idea of extent join. It uses joins over sets to evaluate path expression queries. Its performance depends largely on the number of joins and the size of joining sets. In this section, we will propose some path expression optimizing techniques to reduce the number of joins when evaluating a path expression.

Suppose that path expression /P/E_n be a path beginning from the root element of an XML document tree and if ($\forall$e)(pcpair(p, e) $\in$ Ext(any, E_n)$\neg$adpair(ε, p) $\in$ R(P)), then P is defined as a unique path from root to E_n, written as UP(E_n) = P. Where if P is a path expression, then R(P) means the result set of path P. With the principle of unique path, we get the following two path shorten rules.

Path-shorten rule 1: If UP(E_n) = P, then R(/P/E_n) = Ext(E_n).

Path-shorten rule 2: If P/E/P_2 is a path expression beginning from root and UP(E) = P, then R(P/E/P_2) = R(E/P_2).

Furthermore, let E_1 and E_2 are element names specified in the XML schema or DTD, if ($\forall e_2$)(pcpair(ε,e_2) $\in$ Ext(any, E_2)$\neg$($\exists e_1$)(pcpair(ε, e_1) $\in$ Ext(any, E_1) $\wedge$ adpair(e_1, e_2))), we consider E_1 is a key ancestor of E_2. And let E_1 is a key ancestor of E_2 and there exist path expressions P_1, P_2,..., P_n from E_1 to E_2, if ($\forall e_2$)(pcpair(ε,e_2) $\in$ Ext(any, E_2) $\neg$adpair(ε, e_2) $\in$ $\cup^n_{i=1}$ P(pi_i)), then is called the complementary paths of P_i as for E_1 and E_2($1 \le i \le n$). Then we have the other path optimization rule.

Path complementing rule: If $\cup^n_{i=1}$ (R(P_i)+$\cup^n_{i=1}$ (R(p_i)) is the complementary paths of P_i as for E_1 and E_2, $1 \le i \le n$, then R(P_n) = Ext(E_2) -$\cup^{n-1}_{i=1}$ R(P-i)($\cup^n_{i=1}$ R(P_i) = Ext(E_2)).

As a summary, the path-shorten rules reduce the query costs by shortening the length of path expressions and the path complement rule use the equivalent path expressions to substitute the original ones. Both of them need the knowledge of XML document schema/DTD.

A VISUAL MULTIDIMENSIONAL DOCUMENT EXPLORER

Our system supports three ways for retrieving documents from the database: using query languages, using keywords and explorative browsing. In this study, we describe X-Explorer that provides both keyword based querying and visual multidimensional explorative browsing of the stored documents. Figure 5 is the screen dump of the interface of X-Explorer. The screen is divided into three areas: filtering window, document exploring window and document list window.

The filtering window allows users to select dimensions to explore and enter search conditions. Users can also set up complex Boolean search conditions on the document descriptor in the advanced filtering window. After the search conditions are set up, the system searches for the documents from the database and displays the results both in the document exploring window and the document list window. In the document list window, users can see the values of attributes in the document descriptor of the found documents.

A particular document can be viewed by clicking corresponding entry in the document list window.

The search results are also displayed in the exploring window, where documents are represented as objects in a multidimensional space defined by the chosen attributes. To facilitate visual exploring among large volume of results, the following operations are provided:

- Rotating: Users can rotate the multidimensional space arbitrarily to browse documents from a desired angle.
- Zooming: When the result set consists of large number of document, an object in the multidimensional space might be too small to see clearly. Users can use zooming operations to zoom in/zoom out the space.
- Drilling-down/rolling-up: One point in the exploring window may represent a number of documents with the similar values with respect to the chosen attributes. The drilling-down operation allows users

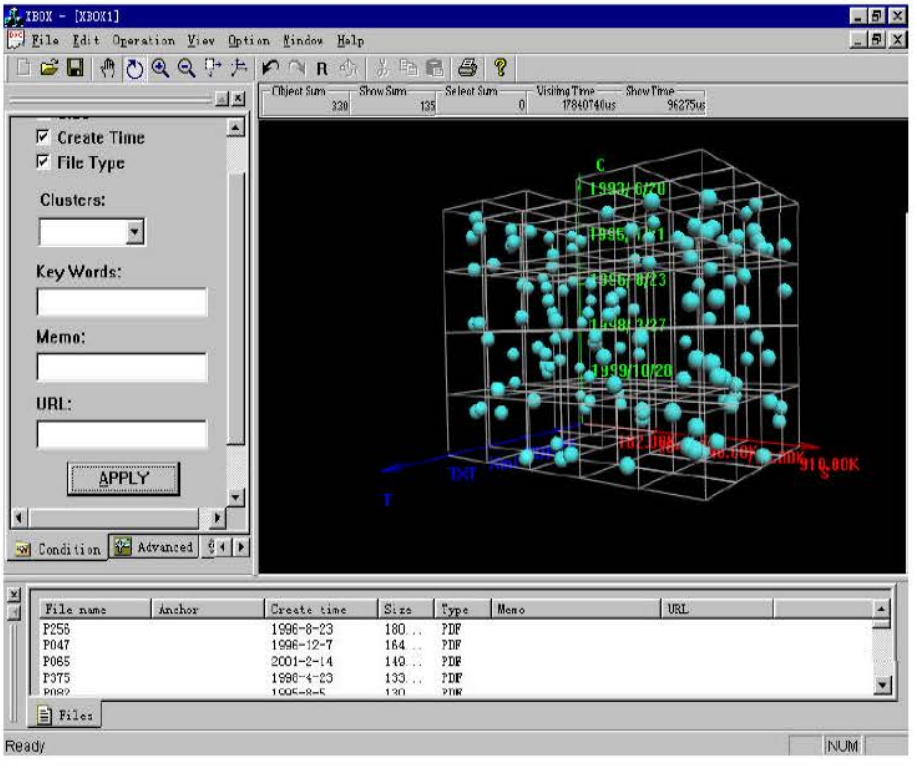

Fig. 5: Interface of X-explorer

to further explore documents contained in the sub-space. Conversely, users can go back to the original space from the drilled down space using the roll up operation.

- Scaling: Users can enlarge or shrink the scale of a given dimension arbitrarily using this operation.
- Moving: This operation allows users to move an axis to change the subspace displayed on the screen. As such, some originally invisible documents can be brought on the screen.
- Undo/redo: These two operations allow user to undo the previous operation or redo the undone operation.

Compared to the existing text-based file explorers and file finding functions available in some operating systems, the X-Explorer has the following advantages.

1. X-Explorer provides more flexible ways to form query conditions, from browsing the full collection to a very specific set of documents. Such a condition can be combinations on both document descriptor attribute values and terms appeared in the documents.
2. Documents are stored in a database. They can be physically clustered based on contents. Both multidimensional index and inverted index are supported. Comparing with file systems where files are scattered around the disks, X-Explorer is able to provide better performance.
3. Most systems support content-based retrieval only on text-based documents. Our system also provides content-based retrieval on non-text based documents, such as postscript files and pdf files.
4. X-Explorer is much more user-friendly than its text-based explorers as a visual-based tool. Users can easily find documents based on his/her vague memory about the documents, such as the time when the document was stored, its approximate size, the type of the document, possible terms contained in the document, etc.

CONCLUSIONS

This study describes a system that manages very large documents based on semantics. Three types of semantics are employed: definitional semantics, associative semantics and schema information. Documents stored in database are physically clustered based on topics. Multidimensional index and inverted index provide efficient access. Structural indexes are implemented to support structure-based XML queries. Two path expression optimization principles are also introduced. A unique visual explorer provides a convenient visual-based explorative browsing of the documents stored in the database.

ACKNOWLEDGMENTS

This research is partially supported by the Teaching and Research Award Programme for Outstanding Young Teachers in Post-Secondary Institutions by the Ministry of Education, China (TRAPOYT) and National Natural Science Foundation of China under grants 60273079 and 60473074.

REFERENCES

1. Carey, M.J., 2001. Towards a scalable infrastructure for advanced E-services. IEEE Data Engineering Bulletin, 24: 12-17.
2. Geffner, S., D. Agrawal and A.E. Abbadi *et al.*, 1999. Browsing large digital library collections using classification hierarchies. Proceedings of the 1999 ACM CIKM Conference, Kansas City, Missouri, USA, pp: 195-201.
3. Gifford, D.K., P. Jouvelot and M.A. Sheldon *et al.*, 1991. Semantic file systems. Proceedings of the Thirteenth ACM Symposium on Operating System Principles, California, USA., pp: 16-25.
4. Vasudevan, V. and P. Pazandak, 1997. Semantic file systems. Technical Report of Object Services and Consulting, Inc. http://www.objs.com/survey /OFSExt.htm, 1997.3.
5. Cark, J. and S. DeRose, 1999. XML Path language (XPath), ver. 1.0. Tech. Report REC- xpath- 19991116, W3C, Nov. 1999.
6. Chamberlin, D., J. Robie and D. Florescu, 2000. Quilt: An XML query language for hetero-geneous data sources. In Proc. Third Int'l Workshop WebDB, Dallas, USA, May 2000.
7. Robie, J., J. Lapp and D. Schach, 1998. XML Query Language (XQL), http://www.w3.org/TandS /QL/QL98/cfp.
8. McHugh, J. and J. Widom. Query optimization for XML. Proc. of the 25th VLDB Conf., Edinburgh, Sept.
9. Goldman, R. and J. Widom, 1997. DataGuides: Enabling query formulation and optimization in semistructured databases. Proc. 23rd VLDB Conf., Athens, Greece.
10. Li, Q. and B. Moon, 2001. Indexing and querying XML Data for regular path expressions. Proc. of the 27th VLDB Conf., Roma, Italy, pp: 361-370.
11. Milo, T. and D. Suciu, 1999. Index structures for path expressions. Proc. Intl. Conf. Database Theory, Jerusalem, Israel, pp: 277-295.

12. Amano, H., Atitsugi and G. Bai *et al.*, 1994. Shusse-Uo: a persistent project of developing a flexible platform for advanced database systems and applications. Technical Report of IEICE, DE93-63. Japan, pp: 137-144.
13. Bai, G. and A. Makinouchi, 1994. WAKASHI/D: A distributed paged-object server for storage management of new generation databases. Proceedings of the International Symposium on ADTI, Nara, Japan, pp: 137-144.
14. Yu, G., K. Kaneko, G. Bai and A. Makinouchi, 1996. Transaction management for a distributed object storage system WAKASHI-design, implementation and performance. Proceedings of the twelfth International Conference on Data Engineering, New Orleans, Louisiana, pp: 460-468.
15. Cattell, R., D. Barry and M. Berler *et al.*, 2000. The Object Data Standard: ODMG 3.0. Morgan Kaufmann Publisher.
16. Zamir, O. and O. Etzioni, 1998. Web document clustering: a feasibility demonstration. Proceedings of 1998 SIGIR Conference, Melbourne, Australia, pp: 46-54.
17. Zhang, T., R. Ramakrishnan and M. Livny, 1996. Brich: an efficient data clustering method for very large databases. Proceedings of the 1996 ACM SIGMOD Conference, Montreal, Canada, pp: 103-114.
18. Guha, S., R. Rastogi and K. Shim, 1999. ROCK: a robust clustering algorithm for categorical attributes. Proceedings of the 15th ICDE Conference, Sydney, Australia, pp: 512-521.
19. Schutze, H. and C. Silverstein, 1997. Projections for efficient document clustering. Proceedings of 1997 SIGIR Conference. Philadelphia PA, USA., pp: 74-81.
20. Gupta, S.K., K.S. Rao and V. Bhatnagar, 1999. K-means clustering algorithm for categorical attributes. Proceedings of First International Conference on Data Warehousing and Knowledge Discovery, Florence, Italy, pp: 203-208.
21. Beckmann, N., H.P. Kriegel and R. Schneider *et al.*, 1990. The R*-Tree: An efficient and robust access method for points and rectangles. Proceedings of the 1990 ACM SIGMOD Conference, Atlantic City, NJ, pp: 322-331.