Modified Element Type ID Representation for XML Structure Retrieval†

Seung-Hoon Jin, Soon-Cheol Kim, Jong-Wan Kim, Sin-Jae Kang

Abstract In this paper, a representation method for structure information retrieval without schema and DTD of XML documents is presented. While existing researches have used DTD or schema to extract structure information, we extract structure information directly from XML documents. Especially the well-formed XML documents are only required to retrieve documents in the proposed method. Thus it can retrieve more general and various documents easily compared to the existing XML retrieval systems requiring DTD or schema. Experimental result indicates that the proposed method retrieves effectively structure information of XML documents independently to DTD information.

Key Words: XML structure retrieval, Schema, DTD.

1. Introduction

With the development of web, the usage of web is growing in various computing environments, and the necessities of efficient storage of web documents and search methods are increasing. In W3C, XML (eXtensible Markup Language) was proposed to overcome weak points and combine advantages of HTML and SGML [1]. In the near future, lots of web information or documents following standard XML format are offered, and it naturally needs a searching method to find necessary information in XML documents.

Since XML documents are written in semi-structured form, the method based on word frequency within documents used in most information retrieval engines is not suitable for XML documents. It is reasonable to support structure retrieval utilizing structure information of XML documents [2]. Most of XML documents have their own DTD (Document Type Definition) or XML schema to declare mark ups to be used in XML documents, to validate the documents, and to achieve document sharing between author and user. XML schema represents shared vocabularies and allows computers to carry out rules made by authors. They provide a means for defining the structure, content and semantics of XML documents. Recently a tool was released by J. Rieger, which translated a DTD into a XML Schema [1]. The translator can map meaningful DTD entities onto XML Schema constructs.

However, we found that many XML documents do not have their own DTD or schema, because there is no problem to display XML documents without them and some people feel a difficulty to write DTD files for complex XML documents. Thus we propose structure information retrieval method not only requiring no additional information such as DTD or schema but also supporting keyword based retrieval and structure based retrieval of XML documents. We can reduce the element search time

† This work was supported by 2000 Research Fund from Daegu University.

significantly by simplifying unnecessary search for entire elements in an element tree generated by DTD.

In the next section, previous related works are described. Section 3 presents a method for XML structure retrieval without DTD. The experiments to test the proposed method are provided in Section 4. Finally, conclusion is followed.

2. Previous research

Previous works to retrieve XML structure information are described. In this research, representative structure representation methods, ETID and LETID methods are presented.

2.1 ETID (Element Type ID)

ETID identifying specific elements and representing hierarchical information between elements is a unique value assigned to each element [3]. After the structure of DTD was logically analyzed, ETID is assigned to each element in DTD. As shown in Figure 1, ETID of section node inherits "/3" of chapter node, its parent node, and then it becomes "/3/1".

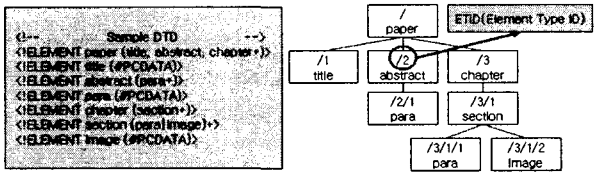


Figure 1. Derivation of ETID from DTD

In Figure 1, an element tree with ETID as a node value is constructed using actual elements and virtual ones. When each ETID value is derived to an identifier of the element tree, a serious recursion problem may be occurred. Recursion is occurred when an element includes element itself like a following example:

<!ELEMENT section(num, title, para, section*)>

Although recursion does not actually exist in XML document, it can be occurred during a step in which an element tree with ETID as a node value is constructed. This recursion is the first weakness of ETID method to extract structure information of XML documents.

When any node like the header node in Figure 2 has alternative components as its internal representation, it is difficult to compose the element tree. To solve the second problem, a method was proposed which modified the process to allocate ETID. The modified method gives an element node ETID by first extracting EID (Element ID) from DTD and then traversing XML documents [4].

```
<!ELEMENT memo (header, body) >
<!ELEMENT header ((to, from, date) |
(time, to, from))>
<!ELEMENT to (heading, name) >
<!ELEMENT from (heading, name) >
```

Figure 2. A case requring selection of node in DTD

Figure 3 shows the process to extract each EID from a DTD file. After allocating EID to all elements, each element inherits EID of its parent node and then each ETID is allocated to represent hierarchical information with traversing XML documents. For example, the EID of class node is "00", member node is a descendant of class node, and the EID of the member node is "01" as shown in Figure 3. Differently from the method [3], there is no virtual node because it assigns ETID to each element by traversing XML documents. The ETID of the member node becomes "/00/01" by combining "00" of parent node with "01" of its own EID [4]. This method extracts only EID without producing hierarchical information with DTD. Therefore, recursion is not happened and it just uses element names in DTD.

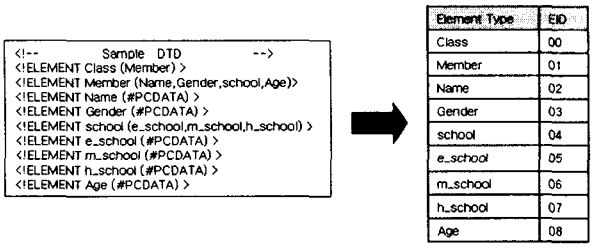


Figure 3. Extraction of EID from DTD

It is possible to assimilate parent-child relationship between elements with ETID information, but not possible to know information for sibling nodes. To know this sibling relation, order information between sibling nodes, SORD (Sibling ORDer) and another order information between same type sibling nodes, SSORD (Same Sibling ORDer) are used [3, 4]. SORD is the occurrence order of elements having same parent and SSORD represents the order between same typed elements having same parent. Figure 4 shows an example XML document represented by ETID method.

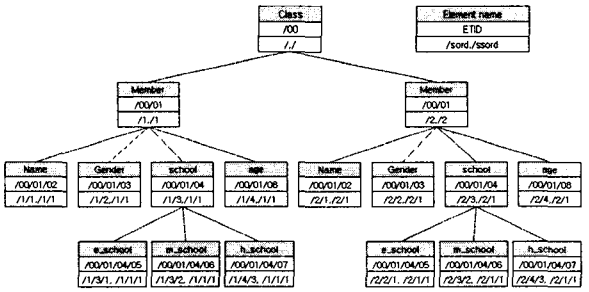


Figure 4. An Example XML Document represented by ETID

2.2 LETID

As the ETID method follows variable length representation, the deeper node extends, the longer its structure information becomes. To reduce this kind of long structure information, the LETID (Leveled Element Type ID) method using 8 bytes fixed-size structure information is proposed [5]. Table 1 shows the meaning of each byte incident in 8 bytes structure information used in LETID.

Table 1. Structure information of LETID

Position	Meaning	Position	Meaning
1,2	Depth of parent node	5,6	Depth of current node
3	SORD of parent node	7	SORD of current node
4	SSORD of parent node	8	SSORD of current node

The information of every element is represented by using 8 bytes, namely higher 4 bytes are used for the information of parent node and lower 4 bytes are for the information of current node. Figure 5 shows the LETID values extracted from DTD.

Some problems happen here. As you have already seen ETID extraction in section 2.1, it is difficult to extract hierarchical information due to recursions can be happened in DTD. Also it is impossible to find out the order between sibling nodes from DTD.

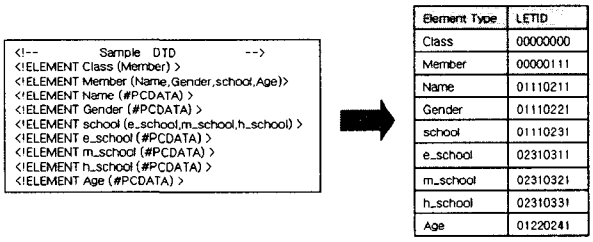


Figure 5. Extraction of LETID From DTD

For example, two XML1 and XML2 files are all valid XML documents according to DTD as shown in Table 2. Since sub section node does not exist in XML1, it cannot assign its SORD and SSORD. However, SORD of the first sub section node is "3" and SSORD is "1", and the second sub section node has SORD of "4" and SSORD of "2" in XML2. As you can see, this information is not provided by the use of just DTD.

Table 2. XML documents according to DTD

DTD <!ELEMENT section(num, title, section*)>	
XML1	**XML2**
<section> <num> ....</num> <title> ....</title> </section>	<section> <num> ..... </num> <title> ..... </title> <section>... </section> <section>...

3. Proposed XML Structure Retrieval

In this section, a method to directly extract structure information from XML documents will be presented. Differently from previous XML structure retrieval methods, only well-formed XML documents are required to obtain the goal.

3.1 METID (Modified Element Type ID)

In this paper, the structure information of XML document is represented by METID method modifying ETID and LETID methods. In the case of LETID method using fixed 8 bytes, there is a limitation to represent the sibling node, because a sibling node and its same named sibling node has 1 byte, respectively. Namely one byte in LETID can represent maximum 62 nodes because it follows single alphabet or digit string like '0'->'9'->'A'->'Z'->'a'->'z' [5]. However, in case of Shakespeare's work [6] used in our experiments, it often happens that the number of nodes is over 62. While the structure information in ETID is represented by inheriting information of parent node, the information quantity increases rapidly according to the depth of tree. In also, the ETID method needs an access to ETID, SORD, and SSORD stored. On the other hand, the proposed METID method follows single string but has no problem over 62 nodes because it supports variable field representation.

Table 3 shows structure information used by METID method. Each field in Table 3 takes basically 1 byte and total 7 bytes are needed. However, we can increase corresponding node to 2 bytes if needed. It means that it is possible to process maximum 62(62 nodes even if we should process an element tree having more than 62 nodes.

Table 3. METID structure information used in the proposed method

Child node Count (CN)	Attribute Count (AC)	Parent node SORD (PS)	Parent node SSORD (PSS)
Current node Depth (CD)	Current node SORD (CS)	Current node SSORD (CSS)	

If we represent the second member node in Figure 4 according to METID method given in Table 3, the node is represented by "/4/0/0/0/1/2/2". Depth of parent node (PD) is not stored substantially in our method. Because CD (current node depth) minus 1 is equal to PD, PD value is easily calculated in program itself without taking a field.

To utilize unoccupied bytes which are taken by LETID but unnecessary in METID, we allocated two bytes for the number of child nodes and the number of attributes in METID. However, if the existing LETID should process more than 62 nodes, it gets into the state of processing impossible as mentioned beforehand.

3.2 METID structure information extraction

To implement the proposed method, we first read XML documents and then created DOM (Document Object Model) objects in Java language [7]. By using DOM, XML document is represented in a tree-structure form and it is possible to traverse nodes by DFS (Depth First Search). We have implemented the system using JAXP reference implementation under Java environment [8]. The reason why we chose DOM APIs is that they were

easy to access each node by transforming XML document to tree-structured object. DOM APIs help us extract easily type and value of each node, and know simply types of node such as Text_Node, Element_Node, and Attribute_Node. As a result, it becomes to extract structure information easily. In case of processing documents in file mode without using DOM, we must locate position of each node with file pointer and type of each node, and then store position information offset of each node [8]. Figure 6 shows the result monitoring the process that it performs structure indexing, content indexing, and attribute indexing by tree traversal after it transformed an XML document on the web to DOM objects.

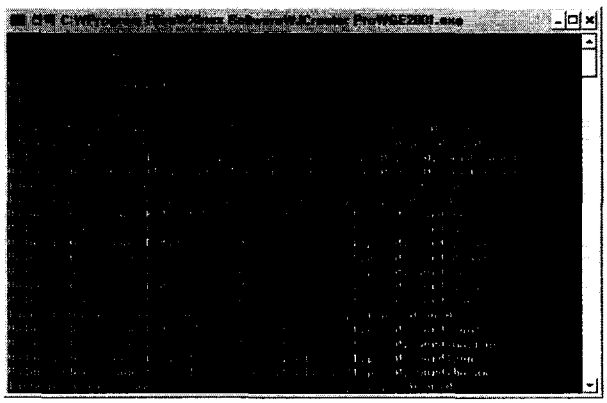


Figure 6. The process to draw structure information

The proposed system manages structure information and index information extracted directly from XML documents in DB. By retrieving DB in which structure indexing, content indexing, and attribute indexing were performed according to the order of queries, it could perform parent node retrieval, children node retrieval, and sibling node retrieval utilizing structure information indexed previously.

3.3 Indexing information

In this paper, indexing information is divided into content index, structure index, and attribute index to show the processing capability of the proposed METID method. Namely, content index is used to do content or keyword retrieval like conventional information retrieval systems. Structure index is applied to retrieve effectively structure information of XML documents. Thus, this information is the essential part of the proposed method. Finally attribute index is additionally chosen to process and retrieve attributes in an element. These indexing information are managed in each other table.

3.3.1 Content index

Content index information is configured based on keywords so as to support contents retrieval. We used Porter's stemmer algorithm [9] to find index word or keyword and exclude stop words.

The index information as shown in Figure 7, is used to process a query like "Find an element or document including hamlet as a content". As the way to find documents or elements including specific content, it makes a SQL statement using input keyword and then shows retrieval result.

keyword	did	elementname	ps	pss	cs	css	cl
hamlet	157	LINE	12	w	6	5	4
hamlet	157	LINE	13	z	5	3	4
hamlet	157	LINE	16	10	1	0	4
hamlet	157	LINE	1b	1U	3	1	4
hamlet	157	LINE	1c	1T	2	1	4
hamlet	157	LINE	1c	1T	5	4	4
hamlet	157	LINE	1c	1T	C	B	4
hamlet	157	LINE	1d	1W	5	4	4

Figure 7. Information for content indexing

3.3.2 Structure index

Structure index information is composed of element, the main component of structure information, DID, METID, and contents. It is the information used to process a query such as "Find the 1st child node of element named speech".

In this research, the ancestor node retrieval, the descendant node retrieval, the sibling node retrieval, the same typed sibling node retrieval, the child node retrieval with specific node count, and the node retrieval with specific attribute count are provided. In the next section, we will present the experiment to find child node retrieval with specific node count.

elementname	did	cs	css	cl	ps	pss	an	text	cn
SPEAKER	156	0	0	4	10	q	0	ADRIANA	0
SPEAKER	156	0	0	4	10	t	0	ADRIANA	0
SPEAKER	156	0	0	4	10	u	0	ANTIPHOLUS OF SYRACUSE	0
SPEAKER	156	0	0	4	10	w	0	ADRIANA	0
SPEAKER	156	0	0	4	10	y	0	ANTIPHOLUS OF EPHESUS	0
SPEAKER	156	0	0	4	11	r	0	Servant	0
SPEAKER	156	0	0	4	11	u	0	PINCH	0
SPEAKER	156	0	0	4	11	v	0	ANGELO	0

Figure 8. Information for structure indexing

3.3.3 Attribute index

Attribute name becomes naturally the essential information for attribute indexing. For mapping with elements to be included attributes, DID and METID are included in the table.

It processes queries such as the node retrieval with specific attribute count and the node retrieval with specific attribute value. To process attribute query is similar to content query processing including a specific keyword.

The following figure shows the information to process query such as "Find the element which has outbound-arrive-time as an element name and year as an attribute name". There is no element having attributes in our test set [6]. So we made an example XML document with attributes and tested the document for attribute indexing.

attributename		elementname			text	did
year		Outbound-arrive-time			1999	160
ps	pss	cs	css	cl	asord	
0	0	3	0	2	4	

Figure 9. Information for attribute indexing

3.4 Retrieval interface

We manage structure information and index information together extracted directly from XML documents. The proposed system retrieves database tables according to the order of keyword or element name in a query in which content index, structure index, and attribute index have been already constructed. Because we can represent structure information in the form of the METID method and utilize it, parent node, child node, and sibling node retrieval could be also performed easily. Figure 10 displays query interface used in the proposed system. For the rest retrieval operations except for child node and parent node retrieval, we could make an SQL sentence by using "AND" boolean operators to be needed after "where" clause. For example, when we select "Element Name" and input "Chapter" as its value, and input "2" for "Sibling Node", an SQL - "Select * from Element where ElementName='Chapter' and SORD='2'" - is made.

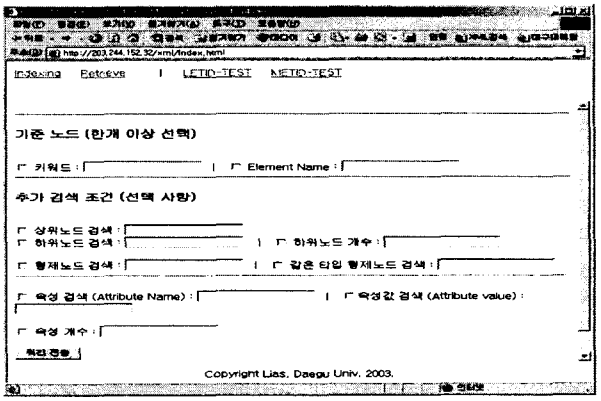


Figure 10. Retrieval interface to make a query

4. Experiments

We performed response time test using a query about a problem finding the number of children nodes. In this experiment, a query "Find the nodes having two child elements among children nodes of the speech element in the test document set" was used to compare the response time of the proposed METID method to that of the LETID method. Total 10 times experiments were performed to the two methods, respectively. So we averaged the response time values for 10 times experiments. To show the superiority of the proposed method, we conducted tests in the way of incrementing the number of test documents. That is, test was performed with one document at first. Next, two documents were provided to the XML structure retrieval system and then the test was conducted with three documents. The experimental results in terms of response time are shown in Table 4.

As you can see in Table 4, the proposed METID method outperforms the existing LETID method in terms of response time for the query finding its children nodes even though we use only three documents. The larger document set we use, the much superior the evaluation results are to the existing method. Especially in accordance with the number of nodes in documents, the response time of the LETID method is directly proportional to the number of nodes in document set. However the proposed method makes little difference in terms of the response time. Due to this reason, there is no problem to show the superiority of the proposed method with only the experiments using small documents.

Table 4. Query response time according to the number of children nodes

# of docs.	# of speech nodes	# of retrieved nodes	Method	
			LETID	METID
1	605	311	20.8sec	0.4sec
2	1743	901	110.7sec	0.6sec
3	2293	1096	183.3sec	0.6sec

Since the document structure of each play in the test set, Shakespeare's work, is same and the element name is same too, we should retrieve lots of elements. There are 78 <ACT> nodes, 2294 <SPEECH> nodes, and 8037 <LINE> nodes in the hamlet.xml document. After indexing total 39 XML documents of 7.58MB size, about 89 seconds were required to retrieve them in the proposed METID method. However, when many queries were presented continuously to DB Server by using the LETID method, the phenomenon that shared memory pool was fully occupied really came about. Therefore out of memory situation was happened under Java programming environment.

On the other hand, it is possible to process queries such as "All nodes having X descendant nodes" and "All nodes having Y attributes" by giving only one query including child node count and attribute count in the proposed METID method. Differently from the METID method, the existing methods using the ETID and the LETID representation must retrieve every descendant node of all nodes stored in DB to process the same query, but it is almost impossible.

5. Conclusion

Many web contents over the Internet tend to be written in the form of XML due to superiority of XML format. XML is well utilized in many fields like e-business, VoiceXML, WML, and so on. In this paper, the proposed method extracts structure information with only XML document using DOM APIs and it performs more simple and efficient retrieval than the existing structure information retrieval algorithm like the LETID method shown in Table 4.

As many people generate web contents easily due to content authoring tools, the needs for efficient document management increase but it becomes difficult to retrieve important documents within reasonable time. Lately Microsoft announced that XML was chosen as its document format for the next MS-Office version. The proposed method offers content index, structure index, and attribute index for supporting efficient retrieval, and therefore it could manage XML documents more efficiently. Since we chose MS SQL, one of conventional RDBs which have been used in many retrieval engines as DBMS, it is expected to be possible that XML content and structure retrieval without significant changes in existing general purposed search engines.

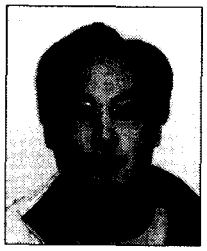
References

[1] W3C, http://www.w3.org/xml/.

[2] Sung-Geun Han, Jeong-Han Son, Jae-Woo Chang, Zong-Cheol Zhoo, "Design and Implementation of a Structured Information Retrieval System for SGML Document", IEEE DASFAA Session 2A: Document Retrieval, pp.81-88, 1999.

[3] Joong Kim, Implementation of XML document

structure retrieval engine, Master Thesis, Honam University, 2001,6.
[4] Hyungil Kang, Yeonggil Choi, Jonsul Lee, Jaesu Yoo, Kihyung Cho, "Design and Implementation of a XML Repository System using RDBMS and IRS," Journal of Korea Information Science Society, Vol.7, No.1, pp.1-11, 2001.
[5] Younki Jo, Jeonggil Jo, Byungryul Lee, Yeonseol Koo, "Representing and retrieving the structured information of XML documents," Journal of Korea Information Processing Society, Vol.8-D, No.4, pp.361-366, 2001.
[6] Jon Bosak, "Shakespeare's work", http://www.guy-murphy.easynet.co.uk.
[7] Seung-Hoon Jin, Indexing and Retrieval System of Structural Information in XML Documents regardless of DTD, Master Thesis, Daegu University, 2003,2.
[8] Yunmyeong Kim, JAVA Programming for XML, Ganamsa, 2002.
[9] William B. Frakes, Ricardo Baeza-Yates, Information Retrieval Data Structure & Algorithms, Prentice-Hall, 1992.

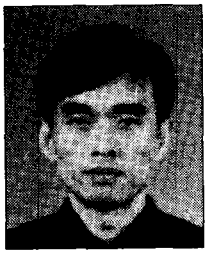


진 승 훈(Seung-Hoon Jin)

2001년 대구대학교 컴퓨터정보공학부 졸업 (학사)
2003년 대구대학교 대학원 컴퓨터정보공학과 졸업(석사)
2003년～호주 어학연수중
관심분야 : 인공지능, 정보검색, XML, 전자상거래

김 순 철 (Soon-Cheol Kim)

1990년 서울대학교 컴퓨터공학과 졸업(학사)
1992년 서울대학교 대학원 컴퓨터공학과 졸업 (공학석사)
1998년 서울대학교 대학원 컴퓨터공학과 졸업 (공학박사)
1998년 서울대학교 컴퓨터신기술공동연구소 특별연구원
1999년～현재 대구대학교 컴퓨터정보공학부 조교수
관심분야 : 운영체제, 분산시스템, 멀티미디어시스템



김 종 완 (Jong-Wan Kim)

1987년 서울대학교 컴퓨터공학과 졸업(학사)
1989년 서울대학교 대학원 컴퓨터 공학과 졸업 (공학석사)
1994년 서울대학교 대학원 컴퓨터공학과 졸업 (공학박사)
1995년～현재 대구대학교 컴퓨터정보공학부 부교수
1999년～2000년 미국 U. of Massachusetts Post Doc.
관심분야 : 지능형에이전트, 퍼지시스템, 인공지능, 정보검색 등

강 신 재 (Sin-Jae Kang)

1995년 경북대학교 컴퓨터공학과 졸업(학사)
1997년 포항공과대학교 대학원 컴퓨터공학과 졸업 (공학석사)
2002년 포항공과대학교 대학원 컴퓨터공학과 졸업 (공학박사)
1997년～1998년 SK Telecom 정보기술연구원 주임연구원
2002년 현재 대구대학교 정보통신공학부 전임강사
관심분야 : 기계번역,정보검색,자동요약,기계학습 등