Generating Nested XML DTD from Non Normalized Relational Views

Mohammed Nasser[1] Hamidah Ibrahim[*]
Ali Mamat[*] Md. Nasir Sulaiman[*]
Faculty of Computer Science and Information Technology, Universiti Putra Malaysia
43400 Serdang, Malaysia

Abstract
Document Type Definition (DTD) is very important because it contains a set of rules that specifies and captures the structure of XML (eXtensible Markup Language) document. Most of the current approaches for generating DTD from relational views do not support nested elements for DTD. Furthermore, these approaches may not generate automatically DTD from flat relational views where they depend on what is submitted by users. This paper investigates how nested XML DTD that support nested elements can be automatically generated from flat relational view. The proposed approach is based on the number of data values for each column in nested view that is generated from flat relational view. The data values for each column of nested view are counted to classify the columns into groups. Each group has the columns with the same number of values. The generated groups represent the DTD elements where the element that has a bigger number of data values is nested into the element that has a smaller number of data values. According to the experimental results, the proposed approach can reduce the storage size of generated DTD by around 31% compared to the other approaches.

Keywords:
flat relational view, nested view, DTD, nested element

1. Introduction

XML has been accepted as the universal standard for data interchange and publication on the web. Because of its flexible syntax, XML allows the same data to be represented in many different ways. Some XML documents may be better designed than other. The need for predefined XML document with DTD is evident in, e.g., data exchange, security views and data integration [2]. The purpose of DTD is to define the structure of a document encoded in XML [6].

Usually, DTD consists of many elements and attributes. The flat relational view is considered as schema but it contains data from multi tables. The columns of nested view are classified into groups based on the number of data values for each column. The mapping of nested relation view into DTD means that each group of columns is defined as an element in DTD. Each column or attribute of the nested view is defined as attribute of elements in DTD.
The rest of this paper is organized as follows. Section 2 presents some definitions about relational views, XML elements and XML attributes. Section 3 presents the related works. Section 4 presents the comparison between DTD and XSD according to the storage size. Section 5 presents and discusses the proposed approach. Section 6 presents the implementation of the proposed approach. Section 7 presents and discusses the results of the experiments. The final section includes the conclusion and the future work.

Manuscript received November 5, 2007
Manuscript revised November 20, 2007

2. Backgrounds

This section presents some definitions for relational views, XML elements, and XML attributes.

2.1 Relational Views

A relational view is a virtual table made up of a subset of the actual tables. A relational view is a named result set of an SQL query. A view allows to display different perspectives of the same database. Views are stored in the database and the view query defines database table's columns and rows that are viewed [10].

2.2 DTD Elements

Elements are the most common form of XML. The first element of XML document must be a root element that must be one. An element can contain other (sub) elements. An element begins with a start-tag and ends with end-tag such as <name> and </name>.
Element content model is the logical structure of the element contents based on the regular expressions such as \?" (0 or 1 instance), *" (0 or many instances), or \+" (1 or many instances) [11]. In the following example, the element paper contains only one instance of sub-element title, one or many instances of sub-element author, and zero or many instances of sub-element citation:
<! ELEMENT paper (title, author+, citation*)>

2.3 Attributes of DTD

Attributes are name-value pairs that contain descriptive information about an element. The attribute is placed

inside the start-tag after the corresponding element name with the attribute value enclosed in quotes [11]. Each attribute declaration has three parts: a name, a type, and an optional default value.
In DTD, elements and attributes are defined by the keywords <!ELEMENT>
and <!ATTLIST>, respectively.
<!ELEMENT> <elem-name> <elem-content-model>
<!ATTLIST> <attr-name> <attr-type> <attr-option>

There are several advantages for elements over attributes, among them [4]:

(i) Elements have order semantics while attribute does not have it.
(ii) Elements can express multiple occurrences better than attributes. Elements and attributes both support a string type.

2.4 Document Type Definition (DTD)

A DTD as in [7] is defined to be $D = (E, A, P, R, r)$
where: E . El is a finite set of elements, A . Att is a finite set of attributes, P is a mapping from $E = \{e \mid e \quad E\}$ to element type definitions, $P(e)$ is defined as the following regular expression : . ::= S | . | e' | . | , . | . * where e' E, “ ” denotes union, “,” denotes concatenation, and “*” denotes Kleene closure, R is a mapping from E to the power set of A. Each $@a$. A can appear in only one $R(e)$. If $@a$. $R(e)$, $@a$ is defined for e, r . E is the element type for the root.

3. Related Works

The work in [1] introduced a prototype that named DTD-Miner. It is an automatic structure mining tool for XML documents. Using a Web-based interface, the mining DTD depends on the user who submits a set of similarity structured XML documents and the tool will suggest a DTD. The DTD-Miner does not support the generation of attribute types and entity references and does not generate a DTD for relational database format.
[2] proposed Transformation Engine for XML (TREX), a middleware system for DTD-conforming XML to XML transformations. TREX is based on the novel notion of XML Transformation Grammar (XTG), which extends a DTD by incorporating XML queries into element type definitions. This allows one to specify how to extract relevant data from a source XML document via the queries, and to construct a target XML document directed by the embedded DTD. TREX efficiently evaluates XTGs by implementing several optimization techniques. XTGs and TREX provide the first systematic method and practical system to support DTD-conforming XML transformations.
[3] presents a frame work for publishing relational database into XML. This frame work provides a language for defining views that are guaranteed to be DTD-conformant, as well as middleware for evaluating these views. It is based on a novel notion of attribute translation grammars (*ATGs*). An ATG extends a DTD by associating semantic rules via SQL queries.
The work in [4] introduces Nested-based Translation (NET) approach which was designed to remedy the problems of FT approach, one need to utilize various element content models of XML. The idea is to find a best element content model that uses “*” or “+ ““using the nest operator. Firstly, the nest operator is defined. Informally, for a table t with a set of columns *C*, nesting on a non-empty column *X* to *C* collects all tuples that agree on the remaining columns C into a set. NeT approach has many limitations among them :(i) It is only applicable to a single table at a time, and cannot obtain a big picture of a relational schema where many tables are interconnected with each other. (ii) It performs only single attribute nesting. Multiple attribute nesting is another interesting research direction. (iii) The translation by using NeT approach will be a flat XML structure [5].
Also there are many tools for generating DTD from either relational database or XML documents such as *Allora*, *DbToXML*, and *Alltova*. These tools generate DTD without considering the nesting of elements. They convert each column of relational view to element or to attribute for DTD based on the user's specifications.

4. DTD vs. XSD for Storage Space

This section presents the comparison between the two XML schemas: Document Type Definition (DTD), and XML Schema Definition (XSD) for the spreading and the storage size.

The elements, attributes and the data types are defined only one time in the body of XML DTD. In contrast, XSD is needed to define the type of data and write the "xsd" or "xd" (based on the type of XSD) before each tag in the schema and the whole of document. This matter leads to a redundancy and costs a high space for storage of XML document. Cleary, XML Schema is too complex to provide even an overview of all its features [16]. According to the comparisons between DTD and XSD in [15], averages of nodes number are 26%, 29% from the nodes for XSD based on number of depth, number of element type, respectively. Rather than, averages of storage size for DTD are 14%, 22% from the storage size for XSD based on the number of depth, and the number of elements type, respectively.

5. The Proposed Method for Generating Nested DTD

This section presents the proposed method for generating DTD that includes elements, attributes and rules. The proposed method is based on the count of data values for columns in nested view. The first step of this method is converting the flat relational view into nested view using functional and multi-valued dependencies. The second step is counting the number of data values for each column of nested view. The third step is classifying the columns into groups based on the same number of data values. The columns that have the same number of data values belong to the same group. The number of DTD elements will be the number of groups. The attributes of each group also will be defined as attributes for related elements of DTD.

5.1 Converting Flat Relational View into Nested View

The flat relational view is converted into nested view to extract DTD. We need nested view for XML DTD for many reasons:

(i) A normal form for nested relational view aims not only to group attributes into related sets of attributes, but also to choose a nested structure with a good representations of the set of semantic connections that already exist in the real world [8].

(ii) The nested relational view is hierarchical structure that is closely related to the hierarchical structure of XML [9].

(iii) The nested normal form is closely related to the XML normal form [9].

The method of converting the flat relational view into nested view is based on the analysis of functional and multi-valued dependencies taking into account the frequency of data values. We assume that the tuples of flat relational view are grouped based on the first column that includes the most frequency of values. Given a flat relational view *FRV, B* is a block that includes a set of tuples based on the frequency of the data values for the first column. Let *n* the number of blocks ***B****, i=1, 2, 3..,n* there for *FRV*= $\bigcup_{i=1}^{n}$ Bi . IF *t* is a tuple $\in B_i$, *m* is a number of tuples for B_i there for $B_i = \bigcup_{j=1}^{m} t_{ij}$ for *j*=1, 2, 3…*m*. Let t_i, t_j are tuples $\in B_i$ for $i \neq j$ and *x, y* are values. If $t_i(x) = t_{i+1}(x)$ *and* $t_i(y) = t_{i+1}(y)$ then remove $t_{i+1}(x)$ *and* $t_{i+1}(y)$ based on functional dependency constraint. If $t_1(x) = t_2(x) = t_3(x)...=t_i(x)$ then remove $t_2(x)$, $t_3(x)$=,…,$t_i(x)$ based on multi-valued dependency constraint.

5.2 Generating Nested DTD from the Nested View

Given a Nested Relational View(*NRV*) from *FRV*, Group *G* is a set of sequenced columns $\subseteq$ *NRV*, *n* is the number of columns in *NRV, i=1, 2, 3,…n.* This implies that nested view *NRV*= $\bigcup_{i=1}^{n}$ G_i. Let *cv* is a function for counting of not null values for each column *cl.* IF $cv(cl_i) = cv(cl_j)$ for $i \neq j$ therefore cl_i, $cl_j \in G_k$ where *k=1, 2, 3,…m.* For each group G_i there is an element E_i for DTD. If *m* is the number of groups, the number of elements is *m*. The attributes of G_i are the attributes of E_i *where* E_i = *att1,att2,…attn.* IF E_i, E_j are two elements $\in$ *DTD,* E_j is nested into E_i where $cv(G_i)$ *of* $E_i > cv(G_j)$ of E_j *for* $i \neq j$.

Example

Given a flat relation view as shown in Fig.1, we want to extract the elements, attributes and rules such that the nesting of elements of XML DTD based on this flat view. Several steps will be done as follows:

CID	Comp-Name	City	OrdID	Order-Date	Pro-ID	Qty	Dis.
ALFKI	Alfreds	Berlin	10643	1-1-2004	28	15	0.25
ALFKI	Alfreds	Berlin	10643	1-1-2004	39	21	0.25
ALFKI	Alfreds	Berlin	10643	1-1-2004	46	2	0.25
ALFKI	Alfreds	Berlin	10952	7-4-2005	6	16	0.05
ALFKI	Alfreds	Berlin	10952	7-4-2005	28	2	0.0
ALFKI	Alfreds	Berlin	10692	15-8-2005	63	20	0.0
ANATR	Ana Tru	México	10759	20-8-2005	32	10	0.0
ANATR	Ana Tru	México	10760	15-9-2005	11	2	0.10
ANATR	Ana Tru	México	10760	15-9-2005	13	10	0.0
ANATR	Ana Tru	México	10760	15-9-2005	19	7	0.20

Fig. 1. Flat relational view for *customers*

Step 1: Convert flat relational view into nested view shown as in Fig.2.

CID	Comp Name	City	OrdID	Order date	Prod ID	Qty	Disco.
ALFKI	Alfreds	Berlin	10643	1-1-2004	28	15	0.25
					39	21	0.25
					46	2	0.25
			10952	7-4-2005	6	16	0.05
					28	2	0.0
			10692	15-8-2005	63	20	0.0
ANATR	Ana Tru	México	10759	20-8-2005	32	10	0.0
			10760	15-9-2005	11	2	0.10
					13	10	0.0
					19	7	0.20

Fig. 2. Nested view for *customers*

Step 2: Count the not null values for each column in the nested view. The result is shown as in Fig. 3.

Cid	Comp Name	City	OrdID	Order Date	ProID	Qty	Disco.
2	2	2	5	5	10	10	10
2			5		10		

Fig.3. Number of values for each column with 3 groups

Step 3: Classify the columns into groups based on the same number of values of the columns as shown in Fig.2. There will be three groups. Group 1 includes three columns *Cid, ComName*, and *City*. Group 2 includes two columns that are *OrderID*, and *OrderDate*. Group 3 includes three columns that are *ProID*, *Qty*, and *Disco.*

Step 4: Create an element of DTD for each group where the elements of DTD will be three elements and each element includes many attributes such that element 1 is Customers that includes *Cid, CompnyName, City* attributes, and element 2 is *Orders* that includes *OrderID, Orderdate* attributes. Thirdly, *Products* represents element 3 that includes *ProID, Qty, and Disco* attributes.

Step 5: Arrange the elements of DTD by considering the element nesting such that the element that has the bigger number of data values is nested into the element that has the smaller number of values. Fig. 4 shows the nesting of elements based on the number of data values.

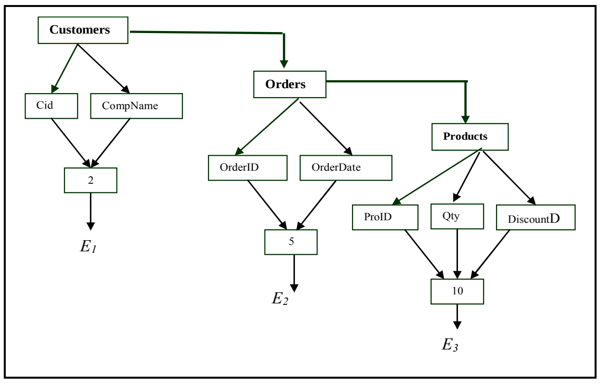

Fig. 4. Nested elements with the attributes

DTD for the above example is as shown in Fig. 5.

```
DocType NVR[
<! ELEMENT NVR(Customers+)>
<!ELEMENT Customers(Orders*)>
   <! ATTLIST Customers
                    CID  ID CDATA#REQUIRED>
                    CompanyName   CDATA #REQUIRED >
                    City CDATA# REQUIRED >
<!ELEMENT Orders(Products*)>
   <! ATTLIST Orders
                       OrderID   CDATA# REQUIRED >
                      OrderDate CDATA #REQUIRED >
 <! ELEMENT Products  Empty>
        <! ATTLIST Products
                         ProID  CDATA# REQUIRED >
                         Qty  CDATA# REQUIRED >
                        Discount  CDATA# REQUIRED >]
```

Fig. 5. DTD for nested view of *customers*

The above DTD contains three elements: *Customers, Orders,* and *Products. Orders* element is nested into *Customers element* where *Customers element has 2 values* whereas *Orders* element has 5 values. *Products* element is nested into *Orders* element because it has data values more than *Orders element.*

5.3 The Algorithm for Generating DTD from Nested View

The algorithm for generating DTD from nested view is shown in Fig. 6.

Input: Nested View
Output: XML DTD
Steps:
1. Count the not null values for each column of nested view.
2. Classify the columns into groups based on the number of not null values such that each group has the columns with same number of not null values.
3. Create an element for each group.
4. Create DTD name with the same name of the nested view.
5. Put the elements of the groups into DTD such that the element for the bigger number of data values is nested into the element for the smaller number of values.
6. Write '+' beside the first element that should be at least one generated element in DTD and write '*' beside the element that has a nested element except the last element that is written 'empty' beside it.
7. Write the attributes for each group into DTD such that each element of DTD is followed by its attributes.

Fig.6. The algorithm for generating DTD from nested view

6. Implementation of the Proposed Algorithm

The proposed algorithm is implemented using Java language via JBuilder6 for enterprise tool with Windows XP. We used *Northwind* database for the experiments and Altova XML SPY 2006 for parsing the output of the proposed algorithm.

7. Experimental Results

In this section, we compare the preliminary results of the proposed approach that is expressed by *PA* with what of other tools such as *Altova*, *Allora*, and *DbtoXML*. This comparison considered 4 samples of views and two metrics of measurement that are the length of DTD and DTD storage size [13, 14] Fig. 7 shows the result of comparisons of DTD lengths in bytes.

XML DTD	***Allora***	***Altova***	***DBto XML***	***PA***
Customers orders products	590	594	488	412
Employees orders	536	510	501	331
Employees orders products	767	762	759	537
Products of orders	693	737	725	393

Fig. 7. DTDs Lengths with different approaches

The results shown in Fig. 7 can be shown graphically in Fig. 8.

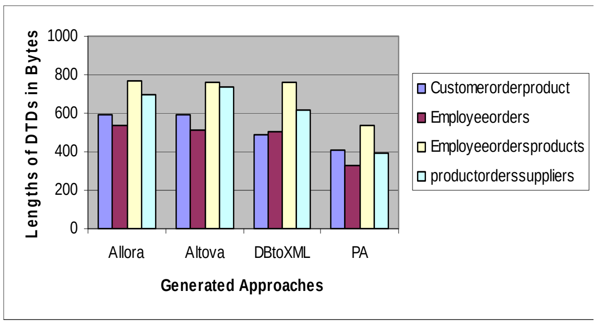

Fig. 8. DTDs lengths for different views in bytes

For the storage size of DTD in bytes, the results are shown in Fig. 9.

XML DTD	***Allora***	***Altova***	***DBtoXML***	***PA***
Customerorderproduct	624	626	519	448
Employeeorders	562	538	531	361
Employeeordersproducts	805	802	801	581
productorderssuppliers	791	775	773	480

Fig. 9. Storage size of DTD for different views in bytes

The results for Fig. 9 are shown graphically in Fig. 10.

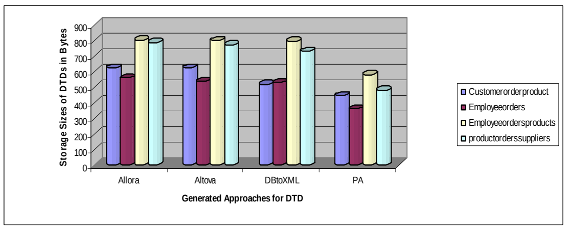

Fig. 10. Storage sizes of DTDs in bytes

Based on the above results of the proposed approach and the other three approaches, the proposed approach reduced the length by 35.42% compared to *Allroa*, 35.49% compared to *Altova*, and 28.74% compared to *DbToXML*. Regarding the storage size, the percentages of the reduction are 32.78%, 31.74%, and 26.87% respectively. It is apparent that the proposed approach generates a nested DTD with lesser length and smaller storage size than that generated by other approaches.

8. Conclusions and Future Work

This paper proposes an approach for generating a nested DTD from flat relational view. This approach is efficient for generating a nested DTD with smaller storage size and lesser length of DTD compared to the other approaches. Using this approach, DTD is generated automatically without any specifications for users. The element model and the attribute model are included together in this approach. It is unlike the other approaches that include either element model or attribute model based on the specifications of users. Also this approach is useful for increasing the semantic of XML documents where it supports the nesting of elements. For future work, we hope to generate nested XML schema instead of DTD from flat relational view.

References

[1] Aoying Zhou, Qing Wang, Zhimao Guo, Xueqing Gong, Shihui Zheng. 'TREX: DTD Conforming XML to XML Transformations'. ACM, 2003.

[2] Chuang-Hue, Moh Ee-Peng Lim ,Wee-Keong Ng. 'DTD-Miner: A Tool for Mining DTD from XML Documents'. IEEE, 2000.

[3] M. Benedikt, C. Y. Chan, W. Fan, R. Rastogi, S. Zheng, and A. Zhou. 'DTD-directed publishing with attribute translation grammars'. VLDB, 2002.

[4] Dongwon Lee, Murali Mani. 'NeT & CoT: Translating Relational Schemas to XML Schemas using Semantic Constraints'. ACM, 2002.

[5] Angela Cristian, Ken Barker, Reda Alhajj. 'Converel: Relationship Conversation to XML Nested Structures'. ACM, 2004.

[6] John W. Shipman. 'Constructing a Document Type Definition (DTD) for XML'. New Mexico Computer Center, 2006.

[7] Wenfei Fan. 'On XML Integrity Constraints in the Presence of DTDs'. ACM, 2001.

[8] Mark A Roth and Henry F Korth. 'Design of 1NF Relational Databases into Nested Normal Form'. ACM Transactions on Database Systems, 1987, pp. 143-159.

[9] Marcelo Arenas. 'Design Principles for XML Data'. PhD dissertation, University of Toronto, 2005.

[10] Ian Gilfillan. 'Introduction to Relational Databases'. Database journal, June 24, 2002.

[11] http://www.rpbourret.com/xml/xmldtd.htm

[12] http://www.XML.com

[13] H. Qureshi, M.H. Samadzadeh. 'Determining the Complexity of XML Documents'. IEEE Conference on Information technology: Coding and Computing (ITCC'o5), IEEE, 2005.

[14] Rebecca J. Cathey, Steven M. Beitzel, Eric C. Jensen, Angelo J. Pilotto. 'Measuring the Scalability of Relationally Mapped Semi structured Queries'. Proceedings of the International Conference on Information Technology: Coding and Computing (ITCC'04), IEEE, 2004.

[15] Erwin Leonardi, Tran T. Hoai, Sourav S. Bhowmick, Sanjay Madria. 'DTD-DIFF: A change detection algorithm for DTDs'. Data & Knowledge Engineering 61 (384–402), 2007.

[16] Chunyan Wang. 'COCALEREX: An Engine for Converting Catalog-based and Legacy Relational Databases into XML'. Master Thesis, The University of Calgary, USA, Nov. 2004.

[17] Irena Mlynkova, Kamil Toman, and Jaroslav Pokorny. 'Statistical Analysis of Real XML Data Collections'. Proceeding of the 13th International Conference on Management of data (COMAD2006), Delhi, India, 2006.