An Efficient Algorithm for Transforming XML Schema into Relational Schema

[1] Abad Shah, [2]Amjad Farooq, [3]Syed Ahsan

[1,3] Al-Khawarizmi Institute of Computer Science, University of Engineering and Technology, Lahore
[2] Department of Computer Science, University of Engineering and Technology, Lahore
Amjadfarooquet@gmail.com

ABSTRACT: The Web and XML have influenced all walks of life especially those that involve business activities over the Internet. People like to do their business activities and transactions from their homes to save time and money. Many business and commercial companies such as insurance companies and banks maintain their records using relational database management systems. But the traditional relational database technology is unable to provide all these new facilities to the users. To enable the traditional relational database technology to cope with the new challenges of the Web and XML technologies, we need a transformation between the XML technology and the relational database technology as a middleware. To achieve this objective, we already proposed and reported an algorithm. In this paper, we extend our previous work and present automation details, testing, and performance report of our proposed algorithm. The result shows that the implementation of the algorithm is more efficient than the existing algorithms for the same purpose [Journal of American Science. 2010;6(11):24-37]. (ISSN: 1545-1003).

Keywords: XML, web, rational database, transforming algorithm

1. Introduction

An electronic document contains regular and irregular structures and may not be completely understood by users (Suciu, 1999; Abiteboul &Vianu, 1997; Brayan, 1997). This type of document (or data) is referred to as *semistructured data (*Suciu, 1999; Abiteboul, 1997). Unlike the data in relational databases (RDBs), the semistructured data is stored without any schema or with a vague schema (Suciu, 1999; Buneman, 1997). There are many other sources of semistructured data, such as the Web, heterogeneous networking of integrated systems, file systems, electronic mail systems, digital libraries etc. (Abiteboul, 1997; Buneman, 1997).

The introduction of Extensible Markup Language (XML) as a standard data/information representation has facilitated the publication of electronic data on the Web (W3C). This language also provides a hierarchical format for the data exchange over the Web with structure (Laurent, 1999; Bray, 2002). Information in a XML document is represented as nested *element* structures (i.e. a tree structure), which start with a root element. An element can have an attribute or a sub-element (for further details about XML see (W3C; Bray et al., 2002). A XML document has an optional part, which is called *Document Type Declaration/Description (DTD).* A DTD of a XML document is considered as the schema of the XML document (W3C; Bray et al., 2002; Men-Hin & Fu, 2001).

A relational database (RDB) has two main components, a schema and a set of operational data files which are created according to the schema. As mentioned earlier, a DTD is considered as a schema of a XML document but there are noticeable differences between a RDB schema and DTD. A complete comparison between them is given in Table 1. The basic difference between them is their structural representations; a DTD represents a hierarchical structure whereas a RDB schema represents a relational (tabular) structure. We can consider XML documents schema analogous to the hierarchical data model schema.

XML is considered as the best tool for representing, transporting and exchanging information on the Web (Laurent, 1999; Bray et al., 2002). This language allows users to define and also display data on the Web. These features make XML powerful and different from Hypertext Markup Language (HTML) (Suciu, 1999; Comer, 2000). XML enables the user to define his own structures using the syntax of the elements in a DTD. A DTD describes the structure of information in a XML document in a hierarchical fashion (Bray et al., 2002). The structure of a DTD consists of elements which are further specified by attributes and/or sub-elements. Recursive and optional type of the sub-element can be defined using the operations * (zero or more times), + (one or more times), ? (optional) and | (or). Many types of data value can be assigned to attributes, i.e. string-type or entity. The data value ANY means that an arbitrary declaration can be made by the programmer. An element in a XML document is uniquely identified by a special attribute ID. This unique attribute of an element can be regarded as the primary key of the element. As it has been mentioned in Table 1, a DTD does not support the concept of the composite ID (or key). An attribute can be referenced in another element through a field called IDREF, and it is a type-less attribute. The concept of an IDREF is similar to the concept of a foreign key in relational databases. There is no concept of a root of a DTD (Bray et al., 2002).

Nowadays, many financial organizations want to empower their customers so that they can perform their financial activities from their homes through the Internet. For these financial organizations to provide their customers with this facility, it is essential and beneficial that the databases (which are mostly relational DB systems) should be presented and processed in the XML format. To provide this facility, we therefore need a technique that can process and transform a RDB and queries into a XML format and vice versa. This technique for the transformation is essential because most of the commercially available database management systems (DBMSs) are relational DBMSs.

To meet these requirements, we have proposed a transformation technique in the form of an algorithm and reported in (Shah et al., 2005). This technique integrates and handles heterogeneous RDBs in the same and uniform manner. Most of investigators agree that the currently available RDB technology alone is not adequate to meet these objectives of using them on the Web without such transformation technique (Shanmug et al., 1999). Recently, some investigators have proposed a few algorithms for this purpose (Shanmug et al., 1999; Men-Hin & Fu, 2001; Williams, 2000; Mani & Lee 2002). In these transformation algorithms, most of the investigators have considered a DTD as a schema of the XML document, and they have used the tree data structure during the transformation. In our proposed algorithm, we didn't use tree data structure because the processes of creating and maintaining tree data structures are expensive and affect the performance of the transformation process as pointed out by Shanmugasundaram et al in (Shanmug et al., 1999). Also, there are many syntax options that are available for writing DTDs. Most of the existing transformation algorithms from DTD into RDB schema are unable to accept DTDs written in different ways (Men-Hin & Fu, 2001; Shanmug et al., 1999). In (Shah et al., 2005), we have used a different approach and proposed a simple algorithm that transforms any DTD of a XML document into RDB schema. In this paper, we extend that work and further report the implementation details and testing results of our algorithm. We also give its analytical analysis and performance comparison with the existing algorithms.

Table 1: Comparison between RDB schema and DTD

RDB Schema	**DTD**
Tabular format	Hierarchical format
It supports many data types.	It supports only character data types.
Schema and tuples of a relation are considered as two different entities, and they are stored in two different files.	XML document and its DTD can be stored in the same file or in two different files.
It is mandatory for a database.	It is optional for a XML document especially for small XML documents.
It is based on the rational data model.	It is not based on any such data model.
It supports the concept of a composite key.	The concept of composite key is not supported.
It supports the concept of foreign key.	Does not support any such concept.
A schema of a relation is defined before creating its instances (or tuples)	Since it is optional, it can, therefore, be defined after the creation of a XML document.

The remainder of this chapter is organized as follows. In Section 2, we describe and analyze the existing approaches for transforming a DTD of a XML document into a relational database schema. In Section 3, we present our proposed approach for transforming a DTD into a relational database schema, and in Section 4, we demonstrate the proposed approach in a case study. Finally, in Section 5, we give our concluding remarks and future direction of this work.

2. Related Work

Investigators have produced many different techniques for transforming DTDs into relational database schemas (Shanmug et al., 1999; Men-Hin & Fu, 2001; Eisenberg & Melton, 2002; Yan, 2001; Williams, 2000; Mani & Lee 2002). There are three (3) main issues that need to be handled during this transformation. These issues are: i) the complexity of the DTD element specifications, ii) the resolution of the conflict between arbitrary nesting in a DTD and relational schema, iii) set-valued attributes and recursion (Shanmug et al., 1999). In the following paragraphs, we give a brief description of the works of these investigators and give, in Table 2 (at the end of this chapter), a comparison of these transformation approaches and our proposed approach.

Shanmugasundaram et al initially proposed an approach in the form of algorithms for transforming a DTD of a XML document into a RDB schema. Men-Hin and Fu later proposed an improvement to the algorithms, (Men-Hin & Fu, 2001; Shanmug et al., 1999). Men-Hin and Fu proposed two algorithms both of which work in the same way, except that they differ mainly in their Step 2 and Step 5. In Step 2 of the improved algorithm they gave more rules for determining the roots. The transformation algorithm by Men-Hin and Fu works in six (6) steps, and they are briefly given below:

Step 1: Simplifying the DTD: This step simplifies DTDs of XML documents using the rules similar to regular expression rules. The information that is useful in constructing schema prototype trees is preserved in the DTDs. The value types (e.g. #IMPLIED, #FIXED etc) for the character data (CDATA) are removed from the DTDs.

Step 2: Determining the Root node: In this step, roots of the prototype trees are determined from the simplified DTDs using the set of rules that are suggested for this purpose.

Step 3: Constructing Schema Prototype Trees: The prototype trees are constructed from the roots that are determined in the previous step using a set of rules.

Step 4: Generating a Relational Schema Prototype: This step realizes a prototype relational database schema from the prototype tree using the following rules:

i) Regard all the necessary attributes and elements in the simplified DTD as the attributes that are treated in the entity- relationship (ER) Model.

ii) Inline all the necessary descendants of the schema prototype tree starting from the root. The necessary descendants refer to all the leaf nodes in the schema prototype tree, and the nodes marked with a “ # ” if they exist.

Step 5: Discovering Functional Dependencies (FDs) and Candidate Keys: In this step the traditional relational database design techniques are applied in order to produce suitable relational schemas. These design techniques reduce the redundancy and inconsistency in a relational database schema, and discover the functional dependencies and the candidate keys by analyzing the newly constructed relational database schema.

Step 6: Normalizing the Relational Schema Prototypes: This step applies the normalization rules on the relational database schema, after determining the FDs and candidate keys of a relational database schema in the previous step.

In the first algorithm of Men-Hin and Fu, hence functional dependencies are found in Step 5, first by analyzing the XML data, and then by applying the algorithm: Efficient discovery of functional and approximate dependencies using partitioning. Step 6 of this algorithm is time-consuming according to Men-Hin and Fu. Hence they modified this step to make the first algorithm more efficient (Men-Hin & Fu, 2001). The modified algorithm decomposes a DTD into small prototypes in Step 4: Tree Construction, and Step 5: Generating a Relational Schema Prototype. The reason for the decomposition is to minimize the cost of finding functional dependencies.

Both of these algorithms-the first and the modified algorithms- use the tree data structure in their transformation processes (or algorithms). The use of this data structure affects the performance of the transformation process because creating and maintaining the tree structure are costly procedures. Also, these two algorithms are unable to handle all types of DTDs as it has been mentioned in (Shanmug et al., 1999).

Eisenberg, and Melton in (Eisenberg & Melton, 2001; Eisenberg & Melton, 2002) gave an informal proposal for a bi-directional transformation between a XML document and a RDB. This transformation can do a complete or a partial transformation at schema level as well as tuple-level (or row-level). The partial transformation may however miss some semantics. This draft for the bi-directional transformations also suggests a transformation of the data types. The authors did not give any proper formal algorithm for these transformations. It is therefore difficult to comment about the real effectiveness of these transformations.

Williams et al have proposed 18 steps for transforming DTD into a relational database schema and 11 steps for the reverse transformation (Williams, 2000). Both of these transformations do not use the tree data structure, but some steps in both of these transformations are unclear. For example, in Step 9 and Step 13 of the transformation of a DTD into a relational database schema, data types are assigned to attributes of a DTD without any explanation and justification. This type of vagueness in the transformation processes makes them difficult to understand and to draw any conclusion about their correctness and accuracy.

Mani, & Lee (Mani & Lee 2002) have proposed a process for transforming a DTD into a relational database schema using a regular tree grammar. The use of the regular tree grammar is helpful in maintaining semantics constraints in the transformation process. The theory of regular tree grammars provides a useful formal framework for understanding various aspects of XML schema languages. The two normal forms (NF1 and NF2) are used for the representation of regular tree grammars. NF1 is used in the XML document validation process, and to check whether a given XML schema satisfies the structural constraints imposed by the schema languages.

XML schema (or DTD) provides several unique data modeling features such as union type "+", which does not exist in the traditional database models such as relational database model. In (Mani & Lee 2002), NF2 is used for representing the basic items of the conversion definition of the two schema languages, that is, a schema that supports union types (e.g., XML-Schema Language (Thompson, 2001), and a schema language that does not support union types (e.g., SQL). This conversion definition is used as the first step in this transformation process of XML schema into relational database schema. The entities and relationships, which form the basic items of data modeling, are represented as elements and attributes of a DTD.

The process of mapping XML schema (or DTD) into RDB schema has several issues, and they have been pointed out in (Mani & Lee 2002). One of the most important among them is the semantic constraint which exists in the XML model. Since relational database schema cannot express these constraints in the XML schema languages, a useful and meaningful subset of those constraints should therefore be found in the mapping process. This process of finding the subset needs simplification of a XML schema. The concept of inlining technique is used for generating an efficient relational schema (Mani & Lee 2002), however; the inline technique that is presented in this work generates a huge number of relations. In addition, this work does not present any proposal for assigning data types to attributes of tables after or during the transformation process.

The transformation process of a XML DTD to relational data schema is the mapping of each element in the DTD to a relation, and it maps the attributes of an element to the attributes of the relation. However, there is no correspondence between elements and attributes of DTDs and entities and attributes of ER model. The attributes in an ER model are often represented as elements in a DTD.

```
<!ELEMENT author (name, address)>
<!ATTLIST author id  ID   #REQUIRED>
<!ELEMENT name (firstname , lastname)>
<!ELEMENT firstname  (#PCDATA)>
<!ELEMENT lastname (#PCDATA)>
<!ELEMENT address  ANY>
```

In the ER model, *author* would be taken as an entity and *firstname, lastname* and *address* would be taken as the attributes of the entity. But in defining a DTD there is no incentive to consider *author* as an element and *firstname, lastname,* and *address* as attributes. In the syntax of a DTD, if *firstname* and *lastname* were defined as attributes, then they could not be nested under name because DTD attributes cannot have a nested structure. A direct mapping of elements to relations therefore leads to an excessive fragmentation.

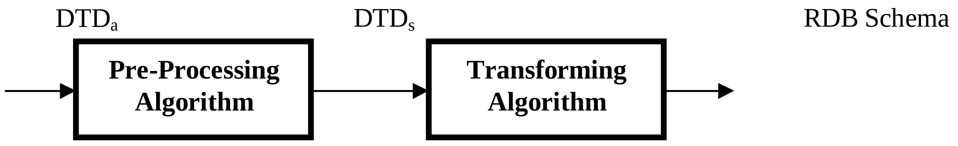

Figure 1: General view of the algorithm

3. The Proposed Algorithm

As mentioned earlier in (Shah et al., 2005), we have reported a transforming algorithm that works in two steps. In the first step it takes a DTD written in any form using the DTD syntax and transforms it into a *standard* form that is devised keeping in view the syntax of a relational database schema. The second step of the algorithm takes the standard DTD and transforms it into a relational database schema. Note that in our approach the transforming algorithm development, we did not include the processes of finding functional dependencies (FDs), assigning data types to attributes of relations after the transformation (note that DTD supports only one data type, i.e. PCDATA), and normalization of the relation database schema. The reason for this is that these three processes are database design techniques and they depend on the perception of a database designer about the database schema (Elmasri & Navathe, 2000). These processes also need direct or indirect expert's intervention and decision. Therefore, in the proposed transforming algorithm, we have separated these three processes from the actual transformation process. Our decision is to separate the manual processes and the automated processes, and this has made our algorithm simpler and helped in achieving our objective of transforming any DTD into a relational schema. In this section, we present only those processes of the transforming algorithm that can be automated.

Table 2: Comparative study of our algorithm and existing algorithms

	BI	**SI & HI**	**GSE**	**DSE**	**WIL**	**RTG**	**EIS**	**OPA**
Data Structure Used	Graph	graph	Tree	Tree	relational structure	regular tree grammars	relational structure	no such abstract data structure is used
Type of conversion	Structured	structured	Structured	Structured	Structured	structural &subset of semantics	mapping	Structural & semantic
Operators Handling	creates a relation for every element in the DTD	ensure that each element is represented only once in a relation	eliminates operators from DTD	preserves some operators to preserve some sub-elements occurrences	some rules are specified to handle them	support XML-Schema (not DTD)	support XML-Schema (not DTD)	Pre-processing algorithm processes them
Advantage	handles fragmentation problem*	the common elements are shared	actual data fields are available in relational schema	number of attributes of the schema is less than the algorithms basic inlining	preserves entities and definitions	maintains semantics constrains	bi-directional transformation	simple, direct mapping & maintains the semantics
Disadvs. F	creates large number of relations	large number of joins in mapping for particular elements	number of possible minimal dependencies is exponential	works with limited number of elements and attributes	some rules of the mapping are vague such as assigning	a complex mapping process	miss some semantics	data types assigning is with human intervention
Prfce. FF	low	low	Low	low	didn't mention	didn't mention	didn't mention	high (expected)

Table Legends:
BI: Basic Inlining (Shanmug et al., 1999), **SI & HI**: Shared Inlining & Hybrid Inlining (Shanmug et al., 1999), **GSE**: Global Schema Extraction (Men-Hin & Fu, 2001), **DSE**: DTD-splitting Extraction (Men-Hin & Fu, 2001) **WIL**: (Williams, 2000), **RTG**: Regular Tree Grammar (Mani & Lee 2002), **EIS**: , **OPA**: Our Proposed Algorithm (Shah et al, 2005).
F: Disadvantages.
FF: Performance.
*: A direct mapping of elements to relations leads to excessive fragmentation of attributes (for more details see [(Mani & Lee 2002).

Our proposed transforming algorithm further consists of two algorithms (or steps): i) *Pre-Processing Algorithm*, ii) *Transformation Algorithm.* In Figure 1, we have given a general view of our transformation process. Pre-Processing Algorithm transforms any DTD that is written in any form into the standard DTD that is referred to as *DTD*$_s$ (see Figure 1). The main objective of Pre-Processing Algorithm is to transform a DTD into a simple uniform and standard form which is denoted as DTDs in Figure 1. The second algorithm, (i.e., Transformation Algorithm), transforms a DTDs in this standard form into a RDB schema (see Figure 1). In the next two paragraphs, we briefly describe the working of these two algorithms. The details of these algorithms can be seen in (Shah et al, 2005).

The main function of Pre-processing Algorithm is to enable the overall transformation process to handle DTDs which are written in different ways, and to transform them into a uniform and standard form. The output of this algorithm is the standard DTD denoted as *DTDs* (Shah et al, 2005) and it is used as the input to Transformation Algorithm as shown in Figure 1. The working of Pre-Processing Algorithm is given in Figure I-1 for more details see (Shah et al, 2005).

Transformation Algorithm takes the DTD$_s$ of a XML document input and transforms it into a relational database schema (RDB_Schema). In Figure I-2, we give the working of the algorithm. In this algorithm, there are two nested loops. The outer loop deals with elements of the DTD$_s$ and transforms them into corresponding tables/relations. The inner loop transforms every attribute of the element into the attributes of the corresponding relation. In Step (iii) of the algorithm (see Figure I-2), it transforms ID and IDREF attributes of an element into primary key and foreign key of the relation, respectively. Note that since the syntax of DTD does not allow the concepts of composite key, therefore, our proposed transformation process also does not support this concept.

1. Implementation Details

As we know from the previous section that the proposed transformation process consists of the two algorithms, and these algorithms are implemented on the platform Intel Pentium II, 400 MHz, Microsoft Windows 98, and using Visual C++ version 6.0. The implementation has two (2) modules corresponding to the two algorithms, namely the Pre-processing module and the Transformation modules. The design of the process is an object-oriented design and it consists of five base classes as show in Figure 2. The designs (or contents) of the classes are given in Appendix II.

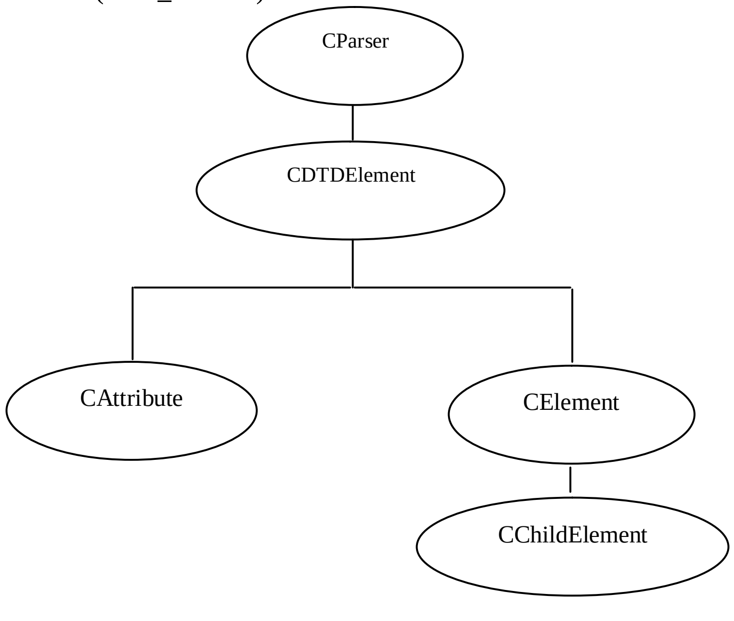

Figure 2: Class-lattice of the transformation process

5. Testing and Analysis

In this section, we give description of the selected test cases and the test data (or instances) which are used to test our proposed transformation process/algorithm. Then we analyze the test results, and based on these results we compare our proposed algorithm with the existing transformation algorithms. We have taken five (5) different and typical test cases and one test data for each test case to test the proposed algorithm. In the next five sections, we describe the five test cases, give their test data, and report their test results.

Test Case 1: Simple DTD

We consider DTD as a *simple DTD* if it contains the basic components/features of the DTD syntax. It does not contain the features such as nesting structures, ID and IDREF(s) attributes, or any referencing to any external entity. We have taken simple DTD as the first test case to the proposed algorithm. A test data (or instance) of this test case that is used for testing the algorithm, is given in Figure III-1 (see Appendix III). The execution of the algorithm for this test data was successful, and the result was as it was expected. The result/outputs of the test data is given in Figure III-2.

For the test case of the simple DTD, our finding is that its automatic transformation using our proposed algorithm into a relational schema is close to its manual transformation.

Test Case 2: DTD Containing ID and IDREF

The concepts of ID and IDREF in DTD are important from database point of view. To test these concepts, we have taken a test case to test these concepts. For this purpose, we have picked a DTD of a Library system as an instance of this test case and it is shown in Figure III-3. After the execution of the algorithm with the instance (given in Figure III-3), the result/output is shown in Figure III-4. After testing this test case, we noticed that the results of our transformation algorithm and the existing algorithms are identical. Note that most of the existing algorithms are not automated.

Test Case 3: DTD Containing Entity Reference and Operators

In a DTD, an entity referencing is a reference to a content of a named entity whether this entity is referencing to a separate external or an internal location, where the content is given in the declaration of the DTD. In this test case, we have picked a DTD as instance of the test case which has this characteristic. An instance (DTD) of this test case is given in Figure III-5. After the successful executing of the instance, the result is given in Figure III-6. The transformation algorithm has worked successfully for the instance, and it has removed the entity declaration from the Catalog DTD as shown in Figure III-6.

Test Case 4: DTD Containing Multiple Root Elements

Some DTDs may have irregular structure which means that these DTDs are missing their root elements. In other words, such type of DTDs have multiple root elements. Our next test case deals this type of DTDs. An instance of this test case is given in Figure III-7. Note that in the figure *Building* element has two parent elements and they are *owner* and *compound*. This instance was successfully executed, and the result of the transformation algorithm is the same as it was expected and given in Figure III-8.

Test Case 5: DTD with Irregular Structure

Usually to write a DTD, there is no fixed format. In other words, DTD of a XML document can be written in many different ways. A DTD is called an *irregular DTD* if it has one element existing as a sub-element of two different main elements (Shah et al, 2005), an example is the element *person* in the Conference DTD, shown in Figure III-9, person element is a sub-element of the two main elements (*editor and author*). An instance of this test case is given in Figure III-9. After testing the instance, the result is shown in Figure III-10. From this test case an interesting result can be concluded, that is, if root element of a DTD contains an attribute of type ID, then the DTD could be a part of another DTD.

DISCUSSION

In the previous section, we report testing result of our proposed and implemented transforming algorithm. These testing results show that this algorithm works successfully on more different types of DTDs as compared to many existing algorithms which are described in Section 2. Another main feature of our algorithm is its implementation because most of the existing algorithms are not implemented; therefore, it is hard to say about their test results and performance. We can conclude that the performance of our algorithm is better than the existing algorithms because our algorithm does not use the tree data structure during the transforming process. Our algorithm saves the heavy construction and maintenance of the tree structure.

During the testing of the algorithm, we have noted that the number of elements and attributes do not affect the working of the algorithm.

Now we summarize our findings of our algorithm, compare it with the existing algorithms and give our concluding discussion. In Table 2, we present the summary of main features of our proposed and implemented algorithm, and the existing algorithms.

In the three algorithms, i.e., Basic Inlining, Hybrid Inlining, and Shared inlining, the evaluation is based on real DTDs which raise the performance concern as it has been mentioned in (Shanmug et al., 1999). This concern is due to a big number of relations generated by the algorithms. In Global-Schema extraction and DTD-Splitting, it has been pointed out in (Men-Hin & Fu, 2001) to the high cost of finding the functional dependencies because the cost of finding the possible minimal dependencies is exponential due to the number of attributes. The other algorithms did not mention their implementation and test results. As we have mentioned earlier that we have implemented our proposed algorithm and successfully tested it by taking different test cases. Whereas, most of the existing algorithms are not yet automated, and also they use graph or tree data structures during their transformation process (see Table 2). We did not use these data structure in our algorithm, which has caused better performance of our algorithm than the existing algorithms. By comparing other parameters in Table 2, it is obvious that our algorithm shows better results than the existing algorithms.

6. Concluding Remarks and Future Directions

In this paper, we have presented the extension of previous work that was reported in (Shah et al, 2005). Here, we have presented development details, testing results and analysis of our proposed algorithm. The proposed algorithm efficiently transforms the DTDs of a XML document (which are written in different ways using different syntax options) into a relational database schema. This transformation algorithm works in two steps/sub-algorithms: i) Pre-Processing Algorithm, and ii) Transformation Algorithm. The first step transforms DTD into a standard form of DTD which is closer to relational database schema. The second step does the actual transformation.

We have demonstrated and tested the working of our proposed algorithm by taking five different test cases. The results are encouraging. The main factures of our proposed algorithm are that it is simpler and easy to understand, implemented and tested on different types of DTDs, and more efficient than the existing algorithms.

A possible extension of this work can be the reverse-directional transformation (i.e., RDB Schema into XML documents schema). These issues and future directions of this work are following:

(i) Handling of IDREFS: It can be an interesting study to translate the concept of IDREFS into relational paradigm.

(ii) Naming conflict between relations and attributes during the transformations.

(iii) Assigning data types to attributes, because DTD supports only character data type and RDB schema supports multiple data types.

REFERENCES

2. Abiteboul, S. & Vianu, V. (1997). Querying the Web, in: Proceedings of the ICDT.
3. Abiteboul, S. (1997). Querying semi-structured Data, in: in: Proceedings of International, Conference on Database Theory.
4. Brayan, M. (1997). SGML and HTML Explained, 2nd Edition, Addison Wesley, 1997.
5. Buneman, P. (1997). Semi structured Data, in: Proceedings ACM Symposium on Principles of Database Systems, pp 117-121.
6. Bray, T., Paoli & J.C.M. (2002). Sperberg-McQueen, E. M., Extendible Markup Language (XML), Second Edition, Available at: http://www.w3c.org/TR/REC-xml.
7. Comer, D. (2000). The Internet Book, 3rd Edition, Prentice Hall, New Jersey.
8. Eisenberg, A., Melton, J. (2001). SQL/XML and the SQL Information Group of Companies, ACM SIGMOS Record, 30(3), pp 101-108.
9. Eisenberg, A. & Melton, J. (2002). SQL/XML is Making Good Progress, ACM SIGMOS Record, 31(2), pp 101-108.
10. Elmasri, R. & Navathe, S. (2000). Fundamental of Database Systems, Third Edition, The Benjamin/Cumming Publishing Company.
11. Laurent & St. S. (1999). XML A Primer, 2nd Edition, M&T Books, California, 1999.
12. Mani M., D. & Lee (2002). XML to Relational Conversion using Theory Regular Tree Grammars, in: Proceedings of the 28th VLDB Conference, Hong Kong.
13. Men-Hin, Y. & Fu, A. (2001). From XML to Relational Database, in: Proceedings of the 8th International Workshop on Knowledge Representation meets Databases (KRDB2001), Italy.
14. Shanmug, J., Tufte, K., Zhang, C., DeWit, D. & Naughton, J. (1999). Relational Database For Querying XML Documents: Limitations and Opportunities, in: Proceedings of the 25th VLDB Conference, Edinburgh, Scotland, pp 302-314.
15. Suciu, D.(1999). Semi Structured Data and XML, in: Proceedings of ACM SIGMOD, pp 430-441.
16. Shah, A., Adeniyi, J. & Tuwairqi, T. (2005). An Algorithm For Transforming XML Documents Schema into Relational Database Schema, in: Patrick van Bommel (Eds), Transformation of knowledge, information and data: theory and applications, Idea Group Publishing, USA, pp 171-189.
17. Thompson, H. (2001). XML Schema, W3C Working Draft, Available at: http://www.w3.org/XML/Schema.
18. Williams, K.(2000) Professional XML Databases, Wrox Press, UK.
19. W3C, (2003) - World Wide Web Consortium: XML, 2003, Available at: http://www.w3.org/XML.
20. Yan, M. (2001). From XML to Relational Database, M. Phil. Thesis, Chinese University of Hong Kong, Dept. of Computer Science.

Appendix I

```
Algorithm Pre-processing (DTD_a):DTD_s
(i)  Scan DTD_Name in DTD_a and make it DTD_s_Name of DTD_s
      and add <!DOCTYPE DTD_s_Name [ ;
      /* DTD_s is the standard DTD */
(ii) Find all the Root_elements of DTDa;
       /* Root_element is
         the element that
         is not sub-element
         of any other
         element in the DTD
         */
(iii)Scan each Root_element of
      DTDa and make it
      Root_element of DTD_s;
(iv) Find total number of main_elements in the Root_element of
       DTD_a, say that they are n;

       /* n is the total number of main_elements */

(v)  Transform the main_elements into DTD_s as follows

    /* main_element has the following
  features:
    (i) a sub-element of the
    Root_element,(ii)a sub-element of
    another main_element, and/or (iii)
    has at least one sub-element, or
    (iv) has at least one attribute */
    <!ELEMENT Root_element
(main_element_1,..,main_element_n)> ;


 (vi)FOR i= 1 to n
     Find total number of sub-elements in main_element_i
     say they are m;
         /* Root_element could be
         main_element; for example if
         it has at least one sub-
         element or at least one
         attribte */

       /* the sub-element has one of
       the following features (i)it
       has no sub-elements,
       (ii)neither sub-element of any
       other main_element nor sub-
       element of the Root_element,
       and (iii)is not
       a main_element */

      IF m > 0 THEN
           Add <!ELEMENT main_element_i (#PCDATA)> ;
           Add <!ATTLIST main_element_i ;

         For j=1 to m
              Transform sub-element_j into DTD_s as sub-element_j CDATA #FIXED

            END FOR LOOP   /*inner loop*/

        attribute:

           IF main_element_i has an
           attribute of type ID THEN
           Transform it in DTDs as
              attribute_name ID
              #REQUIRED
              ELSEIF main_element_i has an
                  attribute of type
                  IDREF/IDREFS THEN
                  Transform it in DTDs
                  as
                    attribute_name IDREF
              TYPE
     /* TYPE is the type of the
        attribute originally exist in
        DTD_a it could be (#REQUIRED,
        #IMPLIED, #FIXED, or
        defaultvalue) just transform it
        as it is in DTD_a */
                 ELSE Transform any other
              attribute defined in DTD_a,
              into DTD_s as it is defined
              in DTD_a
              add '>'

          ELSE add main_element_i to DTDs as its;
           /* it means m = 0; that is
              for the two cases:
              (i)<!ELEMENT
              main_element_i (#PCDATA)>
              ,and (ii)<!ELEMENT
              main_element_i  EMPTY> */

         GOTO attribute;

         END FOR LOOP  /* outer loop */
(vii)    add ']';
(viii)   RETURN (DTD_S);
(ix)     END Algorithm;
```

Figure I-1: Pre-Processing Algorithm

```
Algorithm Transforming (DTD_s): RDB_Schema
```

```
 (i) Scan Root_element of DTDs make it as
Schema_name
          of RDB_schema;
(ii) Find total number of main_elements
  in DTDs (say they are n);
(iii)IF n = 0 THEN EXIT();
       ELSE
         FOR i = 1 to n
         {
      Transform main_elements into Table_i
         and give it the name of the
         element;
         Find total number of attributes
            in main_element_i (say they are
            m);
        IF m = 0 THEN  i++
         ELSE
         FOR j = 0 to m
                Scan attribute_name_j;
                    IF attribute_name_j is
       of type ID THEN
                       make attribute_name_j
                as primary key of Table_i ;
        ELSEIF attribute_name_j is of type
       IDREF THEN
                make attribute_name_j as a
       foreign key of Table_i ;
                  ELSE make attribute_name_j
  a normal attribte_j of
                              Table_i;}
                END FOR LOOP; /* inner
       loop */
          END FOR LOOP; /* outer loop */
  (iv) RETURN (RDB_Schema);
  (v)  END Algorithm
```

Figure I-2: Transformation Algorithm

Appendix II

Here, due to the space problem we give design of sample classes of the case study.

```
Class name: CParser
 Attributes:
 strPath : Char
 m_strError : Char
 m_bIsValid : BOOL = FALSE
 m_strDocTypeStart : Char
 m_strDocTypeEnd : Char
 m_strDocTypeName : Char

Methods:
1.Function ReadDTDFile(strDTDURL:Char )
:Char;
        x= DTDfile;
     {
            get URL of  x;
            read x;
      };
 2.Function
ManipulateParamEntities(strDTDFile:Char,p
ArrNames:CharArray*=NULL,
pArrVals:CharArray*=NULL):Char
        X = Entity;
       {
         RemoveIGNORE(Char );
         RemoveComments (Char);
        Find x;
         IF  (!x)   exit( );
          Else get x name;
         If (parameter x);
         Then  if  x is external entity
           {
              open the file;
              x = file content;
            }
       If x is external entity {get the
       value}
       X = value;
       Call ManipulateParamEntitis;
       };
   3. Function
 RemoveIGNORE(strDTDURL:Char): Char
       {
            Find "<![";
            Find "]]>";
             Remove inbetween;
        }
   4. Function
 RemoveComments(strDTDURL:Char): Char
       {
            Find "<! -  " ;
             Find " - >" ;
             Remove inbetween;
        }
    5. Function
 ConstructElementMap(strDTDFile:Char):
 Char
         x = Element;
          {
             find (x);
             get name;
           if  name is not exist in the
       map (HashTable)
           Then Construct  DTD  Element
       object;
           Else   (Error);
           }
    6. Function
 BuildChildAttributesArray
 (strDTDFile:Char): BOOL
          {
          For each list of attributes
           {
           Find  "<ATTLIST ";
           Get the Element name of the
       attribute list;
           Find Element in the Map (Hash
       Table);
           If   Element not found
           Display (Error);
           }
          For each attribute in the
    attribute list
        {
        get attribute name;
        Determine attribute type;
        Get default value;
        Construct Transformed
    attribute Expression;
        Put in Element
    ChildAttributeArray;
        }
    Repeat
      }
  7.    Function     SaveOutputFile
(strOutputFileName : Char)
    {
     Write "DOCTYPE";
     For  (m_ElementMap)
      {
             write  "<!ELEMENT";
               If (m_bIsEmpty)
                      write  "EMPTY>";
                  Else write
    "(#PCDATA)";
              For (
    m_arrChildAttribute)
                   get attributes;
                   write "<!ATTLIST";
                    write attribute
    name
      }
     get child elements array of the
    Main Element;
      If child element is sub-element
    and  m_bHasOR =False  Then
      {
            Transform it to attribute
    of type CDATA;
       }
          else  attribute is of type
    IMPLIED;
         }
        }
      }
    8. Function SaveOutputMappingFile
    (strOutputMappingFileName:Char)
    {
      Write Relation_Name = DOCTYPE;
      Write Relation Tables
       Write  "(";
       For (m_ElementMAp)
     {
      read element ;
       If element has no child and no
    attraibutes Then (continue);
       Else write (Element_Name)
      Write ")";
      }
    }

Class name: CDTDElement
 Attributes:
```

```
m_strChildElementsExpression : Char
m_strElementExpression : Char
m_bHasOR : BOOL = FALSE
m_bIsEmpty : BOOL = FALSE

Methods:
      1.  Function
          ChildElementArray(pParser:
          CParser*):BOOL
      CEE= m_strChildElementExpression;
      Or = "|";
      Emp = "EMPTY";
      Any = "ANY";
      PCdata = "#PCDATA";
      {
      For each Element
        {
          get CEE;
          Remove operators;
          Remove "(";
          Scan CEE;
           {
           If (Emp) Then m_bIsEmpty=
      true;
             Continue;
           If (Any or PCdata) continue;
         }
           get the name of each
      elements in CCE
           search for the name at the
      m_ElementMap
           If (name not found) Then
      display Error;
           Exit();
           Get CDTDElement of
      ChildElementArray
           Put CChildElement object in
      m_arrChildElements;
           Continue;
         }
       }
      2.  Function
          CheckAttributeValidity(pParse
          r:CParser*):BOOL

        For each attribute
      {
        If attribute name is repeated
      Then (Error);
       }
```

Appendix III

```
<!DOCTYPE   PEOPLE [
 <!ELEMENT  People (Person )>
<!ELEMENT Person  (Name, Address, PhoneNumber,
FaxNumber, Email, Notes)>
<!ELEMENT  Name (FirstName, MiddleName,
FamilyName, Title )>
<!ELEMENT Address (Street1, Street2, City, State,
Country, ZipCode) >
<!ELEMENT FirstName  (#PCDATA)>
<!ELEMENT MiddleNAme  (#PCDATA)>
<!ELEMENT FamilyName  (#PCDATA)>
<!ELEMENT Title  (#PCDATA)>
<!ELEMENT Street1  (#PCDATA)>
<!ELEMENT Street2  (#PCDATA)>
<!ELEMENT City  (#PCDATA)>
<!ELEMENT State  (#PCDATA)>
<!ELEMENT Country  (#PCDATA)>
<!ELEMENT ZipCode (#PCDATA)>
<!ELEMENT PhoneNumber  (#PCDATA)>
<!ELEMENT FaxNumber  (#PCDATA)>
<!ELEMENT Email (#PCDATA)>
<!ELEMENT Notes  (#PCDATA)>]>
```

Figure III-1: People DTD – test cases of the simple DTD

```
RELATION NAME: PEOPLE
RELATION TABLES: (Person,Name,Address)
TABLE: Person
ATTRIBUTES:
          PhoneNumber (Char)
          FaxNumber (Char)
          Email (Char)
          Notes (Char)
TABLE: Name
ATTRIBUTES:
          FirstName (Char)
          MiddleName (Char)
          FamilyName (Char)
         Title (Char)
 TABLE: Address
ATTRIBUTES:
          Street1 (Char)
          Street2 (Char)
          City (Char)
          State (Char)
          Country (Char)
          ZipCode (Char)
```

Figure III-2: Relational schema of the People DTD$_s$

```
<!ELEMENT   Library       (Books+, Publishers+,
Borrowers+, Loans+)>
<!ELEMENT   Books        (#PCDATA)>
<!ATTLIST    Books        LCNo        ID
#REQUIRED
                          PName       IDREF
#REQUIRED
                          title       CDATA
#FIXED
                          author      CDATA
#FIXED>
<!ELEMENT   Publishers   (#PCDATA)>
<!ATTLIST    Publishers   PName       ID
#REQUIRED
                          PAddr       CDATA
#FIXED
                          PCity       CDATA
#FIXED>
<!ELEMENT   Borrowers   (#PCDATA)>
<!ATTLIST   Borrowers   CardNo       ID
#REQUIRED
                        Name         CDATA
#FIXED
                        Addr         CDATA
#FIXED
                        City         CDATA
#FIXED>
<!ELEMENT   Loans       (#PCDATA)>
<!ATTLIST   Loans       CardNo       ID
#REQUIRED
                        LCNo         IDREF
#REQUIRED
                        Date         CDATA
#FIXED>
```

Figure III-3: DTD of the library system

```
RELATION NAME: LIBRARY
RELATION TABLES:
(BOOKS,PUBLISHER,BORROWERS,LOANS)

TABLE: BOOKS
ATTRIBUTES:
          LCNoB (Char) [Primary Key]
          Pname (Char) [Foreign Key] [Not Null]
          TITLE (Char)
          AUTHOR (Char)
          PName (Char)
TABLE: PUBLISHER
ATTRIBUTES:
          PNAME (Char) [Primary Key]
          PADDR (Char)
          PCITY (Char)
TABLE: BORROWERS
ATTRIBUTES:
          CARDNoB (Char) [Primary Key]
          NAME (Char)
          ADDR (Char)
          CITY (Char)
TABLE: LOANS
ATTRIBUTES:
          CARDNoL (Char) [Primary Key]
          LCNoL (Char) [Foreign Key] [Not Null]
          Date (Char)
```

Figure III-4: RDB schema of Library DTD$_s$

```
<!DOCTYPE CATALOG [
<!ELEMENT CATALOG (PRODUCT+)>
<!ELEMENT PRODUCT (SPECIFICATIONS+,
OPTIONS?, PRICE+, NOTES?)>
<!ELEMENT SPECIFICATIONS (#PCDATA)>
<!ELEMENT OPTIONS (#PCDATA)>
<!ELEMENT PRICE (#PCDATA)>
<!ELEMENT NOTES (#PCDATA)>
<!ELEMENT CATEGORY (#PCDATA)>
<!ELEMENT PARTNUM (#PCDATA)>
<!ELEMENT PLANT (#PCDATA)>
<!ELEMENT INVENTORY (#PCDATA)>
<!ELEMENT SPECIFICATIONS (#PCDATA)>
<!ATTLIST PRODUCT NAME CDATA #IMPLIED>
<!ATTLIST CATEGORY TYPE (HandTool | Table |
Shop-Professional) "HandTool">
<!ATTLIST PARTNUM Num CDATA #IMPLIED>
<!ATTLIST PLANT Branch (Pittsburgh | Milwaukee |
Chicago) "Chicago">
<!ATTLIST INVENTORY Status  (InStock | Backordered |
Discontinued) "InStock">
<!ATTLIST SPECIFICATIONS Weight CDATA
#IMPLIED>
<!ATTLIST OPTIONS Finish (Metal | Polished | Matte)
"Matte">
<!ATTLIST OPTIONS Adapter (Included | Optional |
NotApplicable) "Included">
<!ATTLIST OPTIONS Case (HardShell | Soft |
NotApplicable) "HardShell">
<!ATTLIST PRICE Msrp CDATA #IMPLIED>
<!ATTLIST PRICE WholeSale CDATA #IMPLIED>
<!ATTLIST PRICE Street CDATA #IMPLIED>
<!ATTLIST PRICE Shipping CDATA #IMPLIED>
<!ENTITY AUTHOR "John Doe">
<!ENTITY COMPANY "JD Power Tools, Inc.">
<!ENTITY EMAIL "jd@jd-tools.com">]>
```

Figure III-5: The catalog DTD

```
RELATION NAME: CATALOG

RELATION TABLES: (PRODUCT,SPECIFICATIONS,
OPTIONS,PRICE, CATEGORY, PARTNUM, PLANT,
INVENTORY)

 TABLE: PRODUCT
ATTRIBUTES:
          Name (Char)
          NOTES (Char)

TABLE: SPECIFICATIONS
ATTRIBUTES:
          Weight (Char)

TABLE: OPTIONS
ATTRIBUTES:
          Finish (Char) [Not Null]
          Adaptor (Char)
          Case (Char)

TABLE: PRICE
ATTRIBUTES:
          Msrp (Char)
          WholeSale (Char)
          Street (Char)
          Shipping (Char)

TABLE: CATEGORY
ATTRIBUTES:
          AA (Char)

TABLE: PARTNUM
ATTRIBUTES:
          Type (Char)

TABLE: PLANT
ATTRIBUTES:
          Branch (Char) [Not Null]

    TABLE: INVENTORY
ATTRIBUTES:
       Status (Char)
```

Figure III-6: RDB schema for the catalog DTD$_s$

```
<!DOCTYPE HOUSING [
<!ELEMENT OWNER (BUILDING+)>
<!ELEMENT COMPOUND
(BUILDING+)>
<!ELEMENT BUILDING (ROOM+)>
<!ELEMENT ROOM
(BED*,CHAIR*,(CENTRAL_AC
,(EXT_AC|(FAN,HEATER)+)))?>
<!ELEMENT BED EMPTY>
<!ELEMENT CHAIR EMPTY>
<!ELEMENT CENTRAL_AC (#PCDATA)>
<!ELEMENT EXT_AC (#PCDATA)>
<!ELEMENT FAN (#PCDATA)>
<!ELEMENT HEATER (#PCDATA)>
<!ATTLIST OWNER NAME ID #REQUIRED>
<!ATTLIST OWNER AGE CDATA #IMPLIED>
<!ATTLIST COMPOUND ADDRESS ID
#REQUIRED>
<!ATTLIST BUILDING BUILDING_NO ID
#REQUIRED>
<!ATTLIST ROOM ROOM_NO ID #REQUIRED>
<!ATTLIST CENTRAL_AC ELECTRIC_POWER
CDATA #IMPLIED>
<!ATTLIST CENTRAL_AC HORSE_POWER
CDATA #IMPLIED>
<!ATTLIST EXT_AC ELECTRIC_POWER CDATA
#IMPLIED>
<!ATTLIST EXT_AC HORSE_POWER CDATA
#IMPLIED>
<!ATTLIST FAN ELECTRIC_POWER CDATA
#IMPLIED>
<!ATTLIST FAN SPEED CDATA #IMPLIED>
<!ATTLIST HEATER ELECTRIC_POWER CDATA
#IMPLIED>
```

Figure III-7: Housing DTD

```
RELATION NAME: HOUSING

RELATION TABLES:
(OWNER,COMPOUND,BUILDING,ROOM,CENTRAL_
AC,EXT_AC,FAN,HEATER)
```

```
TABLE: OWNER
ATTRIBUTES:
        NAME (Char) [Primary Key]
        AGE (Char)

TABLE: COMPOUND
ATTRIBUTES:
        ADDRESS (Char) [Primary Key]

TABLE: BUILDING
ATTRIBUTES:
        BUILDING_NO (Char) [Primary Key]

TABLE: ROOM
ATTRIBUTES:
        ROOM_NO (Char) [Primary Key]
        BED (Char)
        CHAIR (Char)

TABLE: CENTRAL_AC
ATTRIBUTES:
        ELECTRIC_POWER (Char)
        HORSE_POWER (Char)

TABLE: EXT_AC
ATTRIBUTES:
        ELECTRIC_POWER (Char)
        HORSE_POWER (Char)

TABLE: FAN
ATTRIBUTES:
        ELECTRIC_POWER (Char)
        SPEED (Char)

TABLE: HEATER
ATTRIBUTES:
        ELECTRIC_POWER (Char)
```

Figure III-8: RDB Schema after the transformation

Figure III-9: DTD of there conference

```
ELATION NAME: Conference

RELATION TABLES:
(conf,date,editor,paper,contact,author,person,name,cite)

TABLE: conf
ATTRIBUTES:
        id (Char) [Primary Key]
        title (Char)

TABLE: date
ATTRIBUTES:
        year (Char) [Not Null]
        mon (Char) [Not Null]
        day (Char) [Not Null]

TABLE: editor
ATTRIBUTES:
        eids (Char) [Foreign Key]

TABLE: paper
ATTRIBUTES:
        id (Char) [Primary Key]
        title (Char)

TABLE: contact
ATTRIBUTES:
        aid (Char) [Foreign Key] [Not Null]

TABLE: author
ATTRIBUTES:
        id (Char) [Primary Key]

TABLE: person
ATTRIBUTES:
        id (Char) [Primary Key]
        email (Char)
        phone (Char)

TABLE: name
ATTRIBUTES:
        first_name (Char)
        last_name (Char)

TABLE: cite
ATTRIBUTES:
        id (Char) [Primary Key]
        format (Char)
```

Figure III-10: RDB schema of the conference

5/5/2010