

# Singapore Management University Institutional Knowledge at Singapore Management University

---

Research Collection School Of Information Systems

School of Information Systems

---

6-2000

## DTD-Miner: A tool for mining DTDs from XML documents


Moh Chuang HUE

Ee Peng LIM

Singapore Management University, [eplim@smu.edu.sg](mailto:eplim@smu.edu.sg)

Wee-Keong NG

Follow this and additional works at: [https://ink.library.smu.edu.sg/sis\\_research](https://ink.library.smu.edu.sg/sis_research)

 Part of the [Databases and Information Systems Commons](#), and the [Numerical Analysis and Scientific Computing Commons](#)

---

### Citation

HUE, Moh Chuang; LIM, Ee Peng; and NG, Wee-Keong. DTD-Miner: A tool for mining DTDs from XML documents. (2000). *Second International Workshop on Advanced Issues of E-Commerce and Web-based Information Systems (WECWIS 2000)*. Research Collection School Of Information Systems.

**Available at:** [https://ink.library.smu.edu.sg/sis\\_research/990](https://ink.library.smu.edu.sg/sis_research/990)

This Conference Paper is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email [libIR@smu.edu.sg](mailto:libIR@smu.edu.sg).

# DTD-Miner: A Tool for Mining DTD from XML Documents

Chuang-Hue Moh      Ee-Peng Lim      Wee-Keong Ng

Center for Advanced Information Systems

School of Applied Science

Nanyang Technological University

Nanyang Avenue, Singapore 639798, SINGAPORE

Email: aseplim@ntu.edu.sg

## Abstract

*XML documents are semistructured and the structure of the documents is embedded in the tags. Although XML documents can be accompanied by a DTD that defines the structure of the documents, the presence of a DTD is not mandatory. The difficulty in deriving the DTD for XML documents lies in the fact that DTDs are of different syntax as XML and that prior knowledge of the structure of the documents is required. In this paper, we introduce DTD-Miner, an automatic structure mining tool for XML documents. Using a Web-based interface, the user will be able to submit a set of similarly structured XML documents and the system will automatically suggest a DTD. The user is also able to further refine the DTD generated to reduce the complexity by relaxing some the rules used in the system.*

## 1 Introduction

### 1.1 Background

The World-Wide Web (WWW) is one of the richest repositories of information in the world today. People no longer use the Web simply for browsing but also for numerous E-Commerce applications like making airline ticket and hotel reservations, and performing banking transactions. Nevertheless, for Web based E-commerce to be successful, one has to overcome the interoperability issues between different E-commerce sites. At present, most of the E-commerce sites present their products and services in Web documents that are not very well structured.

The emergence of the Extensible Markup Language (XML) [7] provides a partial solution to this problem. XML is a structured format for data interchange over the Web. The main difference between HTML and XML (and also

the reason for the uprising of XML over HTML) is the use of tags - while HTML tags are primarily used to describe they way in which a data item is displayed on Web browsers, XML tags describe the data itself. The effect is that XML documents are self-describing and this facilitates the post-processing of Web documents, promising a complete change in the “behavior” of the Web in the near future.

Although the structures for a set of structurally similar XML documents can be found in the Document Type Definition (DTD), there are a few problems associated with the use of DTDs as the “schema” for XML documents:

- For a given set of XML documents, the presence of the DTD is not mandatory i.e., we cannot be sure that the DTD will be available with certainty. Note that XML documents that follow the syntax of the XML specification are considered as *well-formed* XML documents. *Valid* XML documents are *well-formed* XML documents accompanied by a corresponding DTD.
- A majority of the Web documents on the WWW today are still in HTML format and do not have any DTD to describe their structure. Refer to [12] regarding the issues involved in “recycling” HTML documents into XML documents.
- The process of defining a DTD for a given set of XML documents is often a complicated and tedious one. In order to define the DTD for the documents, the user must have a clear idea how these documents are structured. To add to the complexity, DTDs do not follow the same syntax as the XML documents themselves. Hence we predict that quite a good number of Web pages on the WWW will not have any DTDs.

The derivation of DTDs for XML documents is indeed a non-trivial task. Many researchers have already recognized the difficulty in deriving DTDs. In [10] it was proposed that

the document types and structures be specified in the XML syntax itself. Further proposals like Schema for Object-oriented XML (SOX) [8], Document Content Description (DCD) [6] and the more recent XML Schema [4, 5] were also based on the similar concept of describing the structures of XML documents using the XML syntax instead of the DTD syntax.

## 1.2 Objectives

The approach that we have taken to overcome the difficulty in deriving DTDs is to build a tool that will automatically suggest the DTD for a collection of XML documents provided by the user. These XML documents are well formed but do not come with a DTD. The derived DTD will provide an overall schema for the document collection. The assumption that we have made here is that the collection of documents are similarly structured and follows a single naming convention for the tags.

The tool known as **DTD-Miner** has been developed based on our proposed algorithms for structural discovery of XML documents. In our approach, the structure of each instance of the XML document collection is represented as a Document Tree and the overall structure of the collection of XML documents is represented using a Spanning Graph. The proposed algorithm derives the Spanning Graph incrementally as one Document Tree is “merged” into the Spanning Graph one at a time. The process ends when all the Document Trees are merged into the Spanning Graph.

DTD-Miner is further equipped with a frame-based Web user interface that allows XML documents to be uploaded from the client computers and presents the derived DTD in the Web browser. It further allows users to refine and simplify the derived DTDs by adjusting an input parameter known as *maximum repetition factor*.

## 1.3 Paper Outline

In this paper, we describe the design and implementation of DTD-Miner. We will cover its system architecture in Section 2. Section 3 gives a brief overview of the Document Tree and Spanning Graph data structures used to represent the instance structure of an XML document and the overall structure of a collection of these documents respectively. Section 4 briefly discusses the heuristic rules used to derive the DTD from the Spanning Graph and also the relaxation of the generated DTD to reduce its complexity. A first version of the DTD-Miner has been implemented and described in Section 6. In Section 5, we walk through an example of using the DTD-Miner. Section 7 describes some related research before Section 8 concludes the paper.

## 2 System Architecture

The DTD-Miner is made up of various modules as shown in Figure 1.

- **DTD-Miner Web Interface:** The Web Interface allows the user to submit XML files. The Web Interface is also responsible for displaying the generated DTD. In addition, it also allows the user to refine the DTD so as to reduce its complexity using a parameter Maximum Repetition Factor, which represents the maximum number of times a child element may appear in the parent element’s definition in the DTD.
- **Pre-processing Module:** An XML file may contain various types of information such as tags, textual data and comments. The Pre-processing module extracts the XML tags (including attribute names) into intermediate files. Apart from simplifying the mining process, the use of intermediate files also allows the user to iteratively refine the DTD generated without involving the original XML files.
- **DTD Generation Module:** This is the main module responsible for the generation of the DTD for a set of structurally similar XML documents supplied by the user. The DTD Generation module consists of the following sub-modules:
  - **Document Tree Extraction Sub-Module:** The main responsibility of this sub-module is to extract the information contained in the intermediate files generated by the Pre-processor module into a Document Tree data structure.
  - **Spanning Graph Construction Sub-Module:** The Spanning Graph Construction sub-module’s primary function is to construct the Spanning Graph from a set of Document Trees (each representing the structure of one XML document in the collection). At each iteration taken by the sub-module, one Document Tree is “merged” into the Spanning Graph and the final Spanning Graph is obtained when all the Document Trees have been “merged” into the Spanning Graph.
  - **DTD Construction Sub-Module:** The DTD Construction sub-module will derive a DTD (text file) from the Spanning Graph by applying the set of heuristic rules. In addition, this sub-module will also perform relaxation of the DTD generated according to the Maximum Repetition Factor if the parameter is specified by the user.

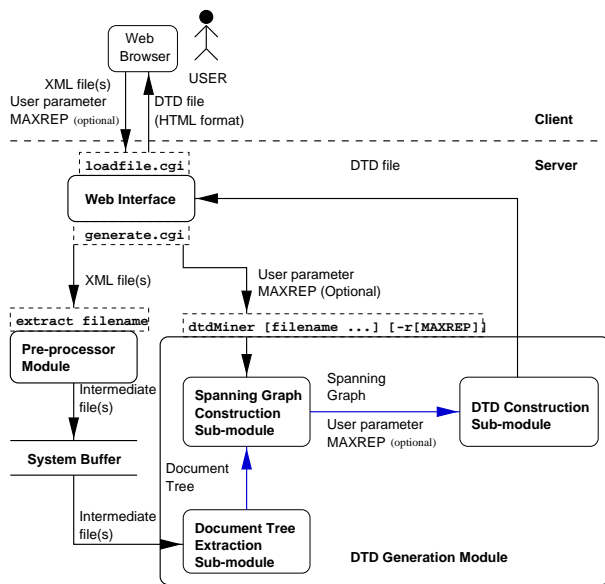


Figure 1. System Architecture Diagram of the DTD-Miner

### 3 Representing Structures Using Document Trees and the Spanning Graph

In this section, we give a very brief overview of the data structures that we have proposed to represent the structure of Web document instances i.e., Document Trees and that to represent the overall structure of a collection of Web documents i.e., the Spanning Graph.

#### 3.1 Document Trees

The structure of XML documents are represented using directed n-ary trees called Document Trees. Each Document Tree is an ordered n-ary tree structure that is uniquely identified by the *Document-ID* (DocID). Each node in the Document Tree is uniquely identified by a system generated *Node-ID* (NID). Note that the *NID* is also unique across all the Document Trees of the collection of structurally similar documents input by the user. The other attributes in each node include:

- **TagName**: Tag name of the document element represented by the node.
- **AttList**: List of attributes of the corresponding document element.
- **PCData**: Boolean variable to indicate whether the corresponding document element contains *Parsed Character Data* (PCDATA).

```
<?xml version="1.0"?>
<!DOCTYPE PLAY SYSTEM "play.dtd">
<PLAY>
<TITLE>All's Well That Ends Well</TITLE>
<FM> <P>Text placed in ...</P>
<P>SGML markup by Jon Bosak, 1992-1994.</P>
<P>XML version by Jon Bosak, 1996-1998.</P>
<P>This work may be freely ...</P> </FM>
<PERSONAE> <TITLE>Dramatis Personae</TITLE>
<PERSONA>KING OF FRANCE</PERSONA>
<PERSONA>DUKE OF FLORENCE</PERSONA>
<PERSONA>BERTRAM, Count of Rousillon.</PERSONA>
<PERSONA>LAFEU, an old lord.</PERSONA>
<PERSONA>PAROLLES, a follower of Bertram.</PERSONA>
<PGROUP> <PERSONA>Steward</PERSONA>
<PERSONA>Clown</PERSONA>
<GRPDESCR>servants to the ...</GRPDESCR> </PGROUP>
... </PERSONAE>
<SCNDESCR>SCENE Rousillon; ...</SCNDESCR>
<PLAYSUBT>ALL'S WELL THAT ENDS WELL</PLAYSUBT>
...
</PLAY>
```

Figure 2. XML Document - "All's Well That Ends Well"

The hierarchical structure present in XML documents can be mapped naturally into the n-ary tree structure of the Document Tree i.e., the parent-child relationship between tree nodes is used to represent the nesting relationship of the document elements. Document Trees are ordered as the ordering information of sub-elements under any parent element will be used to generate the DTD later.

Figures 2 (all\_well.xml) and 3 (win\_tale.xml) show portions of two documents from the XML document collection of "The Life of Henry the Fifth" for the plays "All's Well That Ends Well" and "The Winter's Tale". The Document Trees for these documents are shown in Figures 4 and 5 respectively. Note that each node in the Document Trees is uniquely identified by a NID attribute shown in the nodes of the Document Trees e.g., the node representing <PGROUP> </PGROUP> in the Document Tree of "The Winter's Tale" has a NID of 34 and this is unique across both the Document Trees.

#### 3.2 The Spanning Graph

The Spanning Graph data structure is used to represent the overall structure of a collection of structurally similar XML Document Trees. It can be viewed as an ordered, *directed acyclic graph* (DAG) that encapsulates in it, all the structural information of every Document Tree that it spans i.e., all the the structurally similar Document Trees in the collection. The Spanning Graph is very similar to the Document Tree representation defined for instances of XML documents and is logically similar to the DTD of these XML documents as described below:

- Similar to Document Trees, the edges in the Spanning

```

<?xml version="1.0"?>
<!DOCTYPE PLAY SYSTEM "play.dtd">
<PLAY>
<TITLE>The Winter's Tale</TITLE>
<FM> <P>Text placed in the ...</P>
<P>SGML markup by Jon Bosak, 1992-1994.</P>
<P>XML version by Jon Bosak, 1996-1998.</P>
<P>This work may be freely ...</P> </FM>
<PERSONAE> <TITLE>Dramatis Personae</TITLE>
<PERSONA>LEONTES, king of Sicilia.</PERSONA>
<PERSONA>MAMILLIUS, young prince of Sicilia.</PERSONA>
<PGROUP>
<PERSONA>CAMILLO</PERSONA>
<PERSONA>ANTIGONUS</PERSONA>
<PERSONA>CLEOMENES</PERSONA>
<PERSONA>DION</PERSONA>
<GRPDDESCR>Four Lords of ...</GRPDDESCR> </PGROUP>
...
</PERSONAE>
<SCNDESCR>SCENE Sicilia, and Bohemia.</SCNDESCR>
<PLAYSUBT>THE WINTER'S TALE</PLAYSUBT>
...
</PLAY>

```

Figure 3. XML Document - "The Winter's Tale"

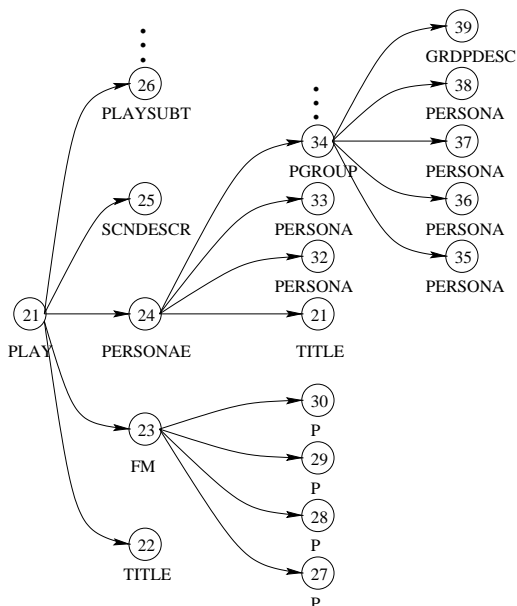


Figure 5. Document Tree for "The Winter's Tale"

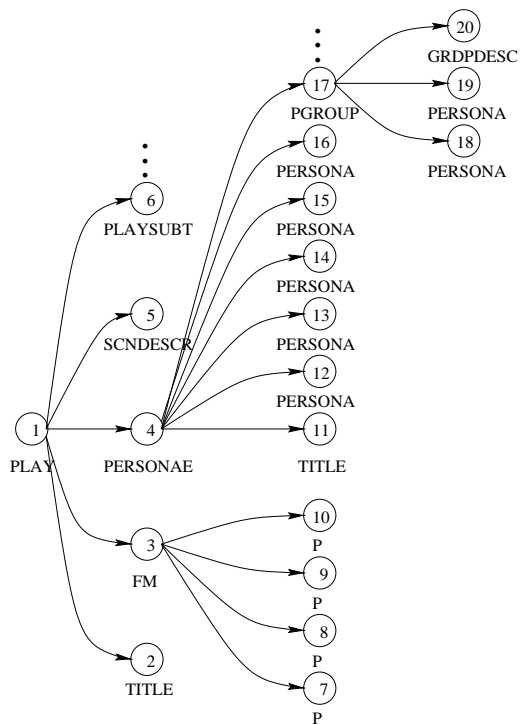


Figure 4. Document Tree for "All's Well That Ends Well"

Graph models the hierarchical relationships between document elements.

- Each node in the Spanning Graph consolidates all information about nodes from the Document Trees sharing the same TagName. In addition, each node contains the following attributes:
  - TagName: Similar to that of the Document Tree, except that each node in the Spanning Graph has a unique TagName.
  - AttList: A list of attributes for the document element. This is a union of all the AttList in the nodes of the Document Trees carrying the same TagName.
  - NodeIDList: A list of nodes from all the Document Trees carrying the same TagName.
- The left-right ordering of sibling edges denotes the left-right ordering of sub-elements of a parent document element. For example, if the node coordinates is the left sibling of the node area in the Spanning Graph and having the two tag pairs as sibling nodes, the tag pairs <coordinates> </coordinates> must appear before <area> </area>.
- Each Spanning Graph edge consolidates information about the parent-child relationship between document

elements found in the Document Trees. Each edge contains an `EdgeIDList` which is similar to the `NodeIDList` of the nodes. The `EdgeIDList` in each edge identifies the parent nodes in the Document Trees in which the parent-child relationship represented by this Spanning Graph edge exists.

Since every node in the Spanning Graph has a unique `TagName`, it uniquely represents an element in the DTD of the XML documents. The number of nodes in the Spanning Graph hence represents the number of elements in the DTD. The edge between two nodes therefore represents a element to sub-element relationship in the DTD. The Spanning Graph representing the combined structure of the XML documents “All’s Well That Ends Well” and “The Winter’s Tale” is shown in Figure 6. In the Spanning Graph, each edge contains an `EdgeIDList`. For instance, the Spanning Graph edge `<PLAY,FM>` has `EdgeIDList = {1, 21}`. The `EdgeIDList` corresponds to the NIDs of the parent nodes in the parent-child relationship between `PLAY` and `FM` in both the Document Trees. Each node in the Spanning Graph has a `NodeIDList`. For example, the node `PLAYSUBT` in the Spanning Graph has a `NodeIDList` of `[6, 26]`. The NIDs corresponds to the nodes in the two Document Trees of with `TagName` of `PLAYSUBT`.

#### 4 Heuristic Rules for DTD Construction & The Relaxation of the DTD Generated

In this section, we outline the heuristic rules that is used to modify the Spanning Graph before generating the DTD. We also discuss the relaxation of the DTD generated.

There are three heuristic rules that are applied to the Spanning Graph for DTD generation:

1. **Define Optionality:** This rule determines whether an element is a mandatory or optional child element of its parent element.
2. **Merge Repeat:** This rule identifies repeating (adjacent) child element of a parent element. Merged elements are either **One-Or-More** or **Zero-Or-More**.
3. **Define Group:** This rules is used to define repeating groups of child elements. Two groups of child elements identified must have identical member elements before they can be merged to form a repeating group. The repeating group of elements can be **One-Or-More** or **Zero-Or-More**.

If the generate DTD has too many rules, it will be too complicated to be of any value to the user. The approach that we have taken to solve this problem is that we allow the user to iteratively relax the generated DTD. The relaxation process is dictated by the Maximum Repetition Factor,

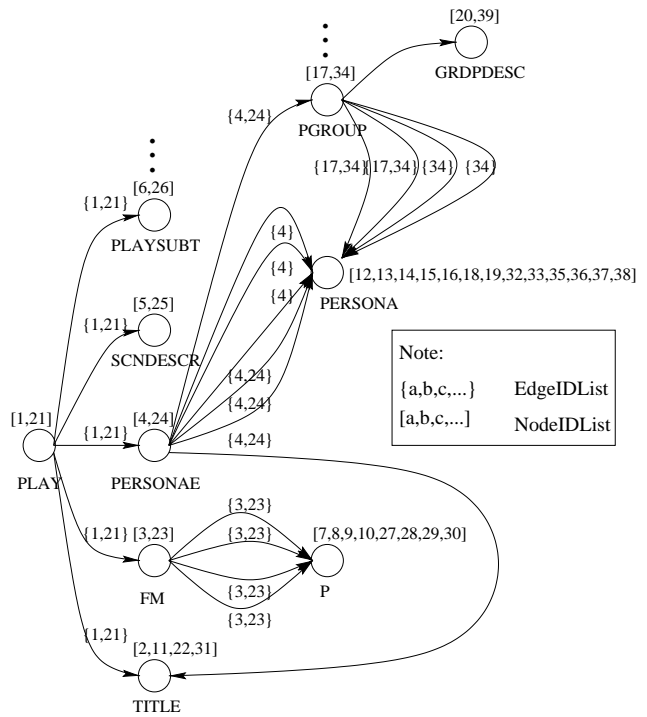


Figure 6. Spanning Graph Constructed for “All’s Well That Ends Well” and “The Winter’s Tale”

which is the maximum number of time a child element can appear in the parent element’s definition. Using this parameter, we repeatedly merge groups of child elements until the number of repetition of all the child elements is less than or equal to the Maximum Repetition Factor.

#### 5 Web User Interface of the DTD-Miner

In this section, we give an illustration of an interaction with the prototype system in order to provide a feel of how the DTD-Miner system works. There are three main screens of the DTD-Miner system: the initial document input screen (Figure 7), the file directory screen (Figure 8) and the DTD display screen (Figure 9).

The system begins at the initial document input screen (Figure 7). The first thing that the user needs to do is to clear away any old files present in the system buffer by clicking on the “Start/Restart” button. The file directory screen (Figure 8) showing no files in the buffer should appear. The user can then submit his own XML documents by first selecting the XML document in his local machine using the “Browse . . .” button and then upload the file to the

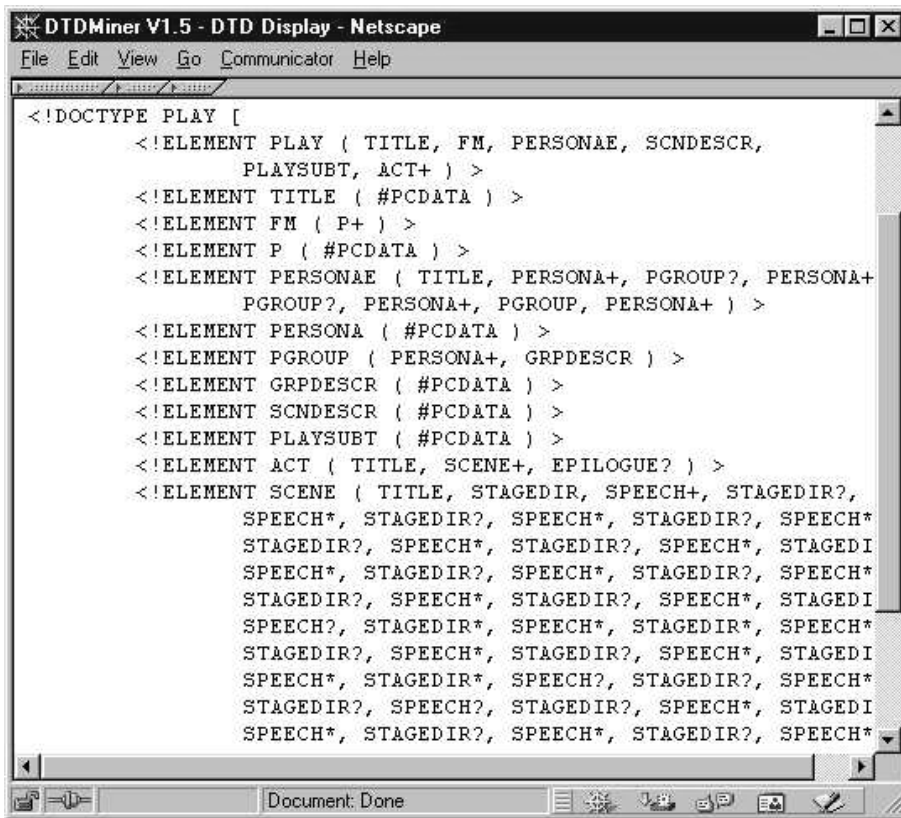


Figure 9. DTD Display Screen



Figure 7. Initial Document Input Screen



Figure 8. File Directory Screen

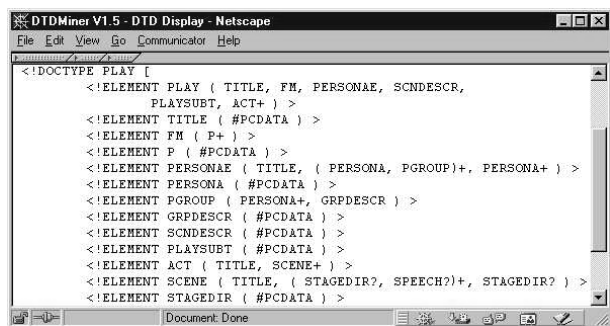


Figure 10. DTD Display Screen (Maximum Repetition Factor of 2)

system buffer using the “Add File” button and the added file should be reflected in the file directory screen. The Maximum Repetition Factor can also be optionally specified here by entering the required value in the text box provided. If this parameter is not specified no relaxation of the DTD will be performed. Finally, the “**Generate DTD!**” will be used to generate a DTD based on the structures of the documents submitted. When the DTD is generated successfully, the file directory screen will be replaced by the DTD display screen (Figure 9).

If the DTD generated need to be relaxed, the user can simply enter the appropriate Maximum Repetition Factor value in the document input screen and click on the “Generate DTD!” button. The re-submission of XML files will not be required. The user can use the “List files” button to bring up the file directory screen and check that all the required XML files are already in the system buffer. The DTD display screen that is shown when we relax the DTD with a Maximum Repetition Factor of 2 is shown in Figure 10.

## 6 Implementation

The implementation of the DTD-Miner is basically separated into the pre-processor program “extract”, the main DTD-Miner program “dtdMiner” and the Web user interface as shown in Figure 1. The programs “extract” and “dtdMiner” are implemented using C++ and Standard Template Library (STL).

The pre-processor program is mainly responsible for extracting the appropriate data into intermediate files. These intermediate files are used by the “dtdMiner” program to generate the DTD instead of the actual XML files.

The Web user interface is implemented using HTML (forms) and CGI scripts (Perl 5). The user input via the

HTML form will be processed by the CGI scripts that will in turn call the DTD-Miner program and supplying the appropriate parameter to the program (with respect to the user inputs). As shown in Figure 1, the Web user interface consist of two CGI scripts, namely “loadfile.cgi” and “generate.cgi”. The script “loadfile.cgi” handles the user input and parameters, include the uploaded XML files. The “generate.cgi” script, on the other hand, is responsible for invoking the “extract” program for pre-processing of the XML files and the invoking the “dtdMiner” program with the appropriate parameters for generating the DTD.

## 7 Related Work

The DTD-Miner System is a prototype for the structural re-engineering framework proposed in [11]. It enables the automatic generation of a DTD from a large set of XML documents while not compromising the ease of use of the user.

The system built by Ashish and Knoblock [3] is one research effort that draws some conceptual resemblance to the DTD-Miner. The system attempts to infer the structure of HTML documents by drawing clues from display and formatting information of the HTML tags and also the indentation to guess the structure of the Web documents. The system however does not deal with XML and DTD.

The structural mining component of the NoDoSE [1] by Adelbreg also draws some similarities to DTD-Miner. NoDoSE is a semi-automatic system for data extraction. NoDoSE is primarily based on plain text files but the HTML Parser component allows HTML files to be handled. There are two main drawbacks of this system for mining structures from XML documents. Firstly, we feel that the degree for user intervention is too extensive and may prove to be tedious for the user when the documents to be parsed are large. Secondly, the NoDoSE does not support XML. The difference between HTML and XML is that in XML documents, the structure of the document can be enclosed within user defined tags.

Another research effort that is closely related to the DTD-Miner is the DTD Generator [9] by Michael Kay. The DTD Generator is a tool that is able to generate the DTD for a given XML document. The main problem with the DTDs generated from the DTD Generator is that it will generate a DTD for every document i.e., the system cannot handle the generation of an overall DTD for a set of structurally similar XML documents. One of the main features of XML is that it allows the user to define their own grammar rules. However, in our opinion, the user would usually define a set of rules for a collection of Web documents rather than a separate set of rules for a single document. This makes the ability for the system to generate an overall DTD essential. It is however,



interesting to note the way the DTD Generator attempts to handle attribute and attribute types.

In [2], an automatic DTD generation tool for SGML documents was described. The work, however, does not address how generated DTDs can be simplified by users using some quantitative measure.

## 8 Conclusion

With the increasing complexities of Web documents and the emergence of XML, user will have a strong need for a tool to automatically extract structures of Web documents. This paper describes the DTD-Miner that is designed for these purposes. The DTD-Miner is a prototype system build from the framework for structural re-engineering XML documents. It has the ability to automatically generate DTD from a set of similarly structured XML documents submitted by the user.

However, the DTD-Miner does not support the generation of attribute types and entity references and does not handle hyperlinks and multimedia data. We are currently looking to further extensions to support attribute types and entity reference generation and also the use of hyperlinks to create inter-document structures. Work is also being done in various methods of simplifying the DTDs generated.

## References

- [1] B. AdelBerg. NoDoSE - A Tool for Semi-Automatically Extracting Semi-Structured Data from Text Documents. In *ACM SIGMOD International Conference on Management of Data*, pages 283–294, 1998.
- [2] H. Ahonen. Automatic generation of sgml content models. In *Electronic Publishing*, pages 195–206, Helsinki, Finland, 1996. Wiley Publishers.
- [3] N. Ashish and C. Knoblock. Wrapper Generation for Semi-structured Internet Sources. *ACM SIGMOD Record*, 26(4):8–15, 1997.
- [4] D. Beech, S. Lawrence, and M. Maloney. XML Schema Part 1: Structures. Technical report, World Wide Web Consortium, <http://www.w3.org/1999/05/06-xmlschema-1>, 1999.
- [5] P. Biron and A. Malhotra. XML Schema Part 2: Datatypes. Technical report, World Wide Web Consortium, <http://www.w3.org/TR/xmlschema-2/>, 1998.
- [6] T. Bray, C. Frankston, and A. Malhotra. Document Content Description for XML. Technical report, World Wide Web Consortium, <http://www.w3.org/TR/1998/NOTE-dcd-19980731.html>, 1998.
- [7] T. Bray, J. Paoli, and C. Sperberg. Extensible Markup Language (XML) 1.0. Technical report, World Wide Web Consortium, <http://www.w3.org/TR/1998/REC-xml-19980210>, 1998.
- [8] M. Fuchs, M. Maloney, and A. Milowski. Schema for Object-oriented XML. Technical report, World Wide Web Consortium, <http://www.w3.org/TR/NOTE-SOX>, 1998.
- [9] M. Kay. SAXON DTD Generator - A Tool to Generate XML DTDs. At <http://home.iclweb.com/icl2/mhkay/dtdgen.html>.
- [10] A. Layman, E. Jung, and E. Maler. XML-Data. Technical report, World Wide Web Consortium, <http://www.w3.org/TR/NOTE-XML-data>, 1998.
- [11] C.-H. Moh, E.-P. Lim, and W.-K. Ng. Re-engineering Structures from Web Documents. Submitted to conference, 2000.
- [12] A. Sahuguet and F. Azavant. Web Ecology: Recycling HTML pages as XML documents using W4F. In *ACM Workshop on the Web and Database (WebDB)*, pages 31–36, 1999.