Abstract. Today’s XML editors provide basic functionality such as creating, editing and parsing documents and only a little information about a given XML element at runtime. Sometimes the user wants to know what valid element can be added without looking in to the DTD or validating the whole document.

This paper presents a tool that solves that problem and was designed to be a link between DTD and XML Editors.

1 Introduction

Extensible Markup Language (XML) is a language widely used to represent electronic data and XML editors try to simplify the handling of creating such documents. XML is a structured language and follows grammar rules that are usually stored in a DTD (Document Type Declaration) or Schema. Common XML Editors use XML Java APIs, such as Xerces, jdom and many more. These implementations provide methods of creating elements and validating documents to a given DTD and handle the IO operations of a XML file. There is no useable interface to extract information from a DTD to a given XML element at runtime.

Example

<table>
<thead>
<tr>
<th>Element abcd as described in the DTD</th>
<th>XML tree generated by an XML editor so far</th>
</tr>
</thead>
<tbody>
<tr>
<td>… &lt;!ELEMENT abcd (a, b, c, d)&gt; …</td>
<td>&lt;abcd&gt; &lt;a&gt; …&lt;/a&gt; &lt;b&gt; …&lt;/b&gt;</td>
</tr>
</tbody>
</table>

The user doesn’t know which valid element can be added after b, in this case c, without looking into the DTD or using the trial & error method, such as adding any element and validating the document with the XML java implementation in use.
XGA can be used to extend existing XML java implementations and work as a link between XML and DTD.

Feature list:

- Static information that is extracted directly from the DTD:
  - all elements declared in the DTD
  - all possible children of a specific element
  - information about attributes, such as default value, type, mode(required, implied or fixed)
- Dynamic information that is processed at runtime for a specific XML element
  - allowed previous/next siblings
  - required/allowed attributes
  - can set an attribute
  - required/allowed first child if element has no children
  - can append another element
  - can be replaced by another element
  - can be removed
- Validation of a single element or subtree of an element

The basis for XGA is the data classes ElementDeclaration and AttributeDeclaration, which are explained in section 2, especially the makeup of the elements content model tree. The design of the tree algorithm is the trickiest part to make the API stable and efficient. We describe the basic functionality of the CM (content model) tree algorithm in section 3. One of the project’s major goals is to achieve independency in the use of a XML Java Implementation. Certain operations have to be implemented by the user, which are fairly simple and straightforward. Section 4 introduces a wrapper interface for XML elements and the primary class ElementInfo of the API. ElementInfo provides basic methods, such as finding the next valid neighbour, required children or required attributes, which can be combined into more complex methods.

2 Data Extraction and Content Model Tree

While the content of a DTD can be characterized as the grammar rules that every instance of a XML file has to follow, it holds information for all declared XML elements, such as number of attributes, possible children and parents, . . . . The main idea of data extraction is to put all the information in an easily readable data structure. With the help of NekoDTD ([2]) (a program which transforms a DTD into a specific XML structure to access the DTD’s content more easily) we have 3 classes that represent the key parts of every DTD

- DTDGrammar as the whole DTD itself
- ElementDeclaration as the equivalent of an XML element
- AttributeDeclaration as the equivalent of an XML elements attribute
2.1 Content Model Tree

The content model is part of the element declaration in an XML document and constrains which elements can be children. Four types of content models are declared in the W3C XML Specifications (3rd Edition).

- **EMPTY** – identifies elements with no content
- **ANY** – any declared elements in the DTD can be children
- **MIXED** – elements and PCDATA (parsed character data, usually text) can be children in a mixed order
- **Children** – a regular expression that defines which elements can be children and in which order they have to appear. The content model includes grammar rules for choice lists, sequence lists and single elements. Occurrences of elements or groups of elements are marked by defined operators. In the next section we bring the regular expression in a computable form in order to perform our algorithm on it.

The model of the data structure, presenting the content model, is an undirected, unweighted and single-edged graph, a so-called simple graph. To simplify matters, we assume that the graph is a tree, where every node can have one parent, up to 2 children and elements as leaves. We differentiate between 5 types of nodes:

- **Root Node** – is the uppermost node in the model and works as an end point for the algorithm. Its child is the first enclosing group node of the content model.

- **Group Node** – has only one child and works as an indicator for grouped elements and its occurrences

- **Choice Node** – left child is always a group or element node and the right one can be the next choice node or the last group/element node in the choice list

- **Colon Node** – same as choice node, just elements are taken from the sequence list

- **Element Node** – end/start point for the algorithm and stores its occurrence
Occurrences are only stored in group or element nodes and are described as follows, according to the XML specifications.

- “1” – one time
- “?” – zero or one time
- “*” – one or many times
- “*” – zero or many times

Example: tree made up from ( ( a, b ) | ( c?, d, e? ) | f+ )*

3 CM Tree Algorithm

The algorithm operating on the content model tree is the core of the XGA project. We build a recursive algorithm, which is dependant on two input parameters, the current direction of the algorithms path (up or down, seen from the tree view) and the actual parent node, to find the next allowed elements beginning from a chosen start element. Different rule sets, according to the current node and the input parameters, determine the next node on the algorithm’s path. Sometimes additional information is needed such as the occurrence of children’s nodes. The algorithm is not restricted to taking only one way and can change direction on occasion. To give an easier way to understand basics of the algorithm we will try to show some of the rules in a graphical way.
Nodes with Colon Nodes as parents:

Occurrence: ?, * or X is start point of the algorithm

Occurrence: 1, +

Occurrence: ?, *

Left Childs occurrence: 1, +

Left Childs occurrence: ?, *

Nodes with Choice Nodes as parents:

We have to make sure that every valid element of a choice sequence is taken in the solution set.
This subset of rules covers the most common cases of how the algorithm decides on the next nodes on its path. With this approach we may experience some problems with specific content model patterns and infinite loops when changing the direction on the path (up/down) for example. We want to introduce 3 additions to the algorithm:

- We add a Tabu List to store already processed elements. We avoid infinite loops when the algorithm changes the direction and reduce repetitive calls of the algorithm’s methods.

- The second addition covers a problem with patterns of a specific form. Simplified, the algorithm works quite iteratively and if it hits the end of a group, it checks its occurrence and if its not + or *, it stops. This leads to some problems with nested groups, where the occurrence of an upper group is crucial to the algorithm’s behaviour.

  o  (a?, b, c, ((d| e), f?, g*, h?))*

  starting point: d
  current point: h
  output so far: f, g, h
  the algorithm only sees the nested group and its occurrence and would stop. The second addition starts here and processes every nested group up to uppermost one and checks if the first element in this group can be reached. In this case, the output will be extended by “a” and “b”.

- Occ. of left child: +, 1 Occ. of left child: ?, *
4 Technical Details

- The ElementWrapper class is the only class that has to be implemented by the user and is a link to the XML editing software and XGA. It provides XGA with required information of the XML elements context (next sibling, parent, ...). To use XGA right from the start, a simple ElementWrapper based on the DOM interfaces from W3C is included in the package and should work with all DOM XML APIs (tested on the implementation from Xerces 2.0 API).

- The ElementInfo class is the primary interface for comparing information taken from the DTD with elements of a XML document. It provides the basic functionality to find neighbours for an XML element, missing attributes, a simple validating algorithm, ... ElementInfo is not instantiated directly, it needs access to an XMLGrammar object to get static information from the underlying DTD and it is linked to an ElementWrapper object that represents the actual element in the XML Document.

5 Related Work

The DTDParse developed from Wutka is the first hit when you are looking for a DTD Parser on the internet (Google). The parser reads a DTD and returns an object, which is similar to the Document Object Model that can be accessed or altered by methods of the DTDParse’s API.

Oracles XML Development Kit (XDK), as one of the first software companies, supports the core parts of the DOM level 3 Validation ([5]) specification published by the W3C. On the W3C site for the DOM level 3 specifications are JAVA interfaces available which serves as guideline for programs or scripts to dynamically update content or structures of a document and to ensure it stays valid. Oracle’s XDK works only with Schemas, but provides a tool to transform DTD into Schema. One weak point in XDK is that you have to use the Parser included in Oracles XML API to use all the features of the DOM validation interfaces, whereas XGA is independent on the XML parser you are using (jdom, xerces, dom4j, ...).
6 Conclusion and Future Work

We have developed a JAVA API that helps the user creating an XML document in supporting him with various information of the underlying DTD, such as finding allowed siblings to a specific element, detailed information about an elements attributes and a validation method for every element or its subtree.

As the DOM level 3 Validation specifications will likely become a standard in the industry and oracle as the only representative we might modify XGA in the future to be consistent with the DOM level 3 Validation interfaces. Many features of the DOM interfaces are already supported by XGA. For a detailed comparison, refer to the DOM interface ElementEditVAL ([6]) and the feature list in our introduction.

Another idea is to implement a semi auto XML element generator for editors using XGA to construct a valid element with its children and attributes.

In the near future, Schema will become more important in the XML world and some future work could be to write an algorithm to transform the Schemas XML based structure of the content model into the CM Tree structure.

References

[1] Tim Bray (Textuality and Netscape), Jean Paoli (Microsoft), C. M. Sperberg-McQueen (W3C), Eve Maler (Sun Microsystems, Inc.), François Yergeau: Extensible Markup Language (XML) 1.0 (Third Edition), http://www.w3.org/TR/2004/REC-xml-20040204/