XML Views for Electronic Editions

Ionut E. Iacob[*]
Department of Computer Science
University of Kentucky
Lexington, KY, USA
eiaco0@cs.uky.edu

Alex Dekhtyar[†]
Department of Computer Science
University of Kentucky
Lexington, KY, USA
dekhtyar@cs.uky.edu

ABSTRACT

In this paper we discuss the implementation of user-defined views over multihierarchical document-centric XML documents.

Categories and Subject Descriptors

I.7.2 [**Document and Text Processing**]: Document Preparation—*format and notation, hypertext/hypermedia, markup languages*

General Terms

Management, Design

Keywords

XML views, editor design

1. INTRODUCTION

Electronic Editions of documents require significant human effort at production time. Human editors must prepare the proper annotation of the document, often with markup from multiple XML hierarchies [2, 3]. In this paper we describe user-defined views over multihierarchical document-centric XML documents and their use in the editorial process (Section 2). XML views allow human editors to prepare and observe, during the editorial process, only portions of the markup relevant to their current task. We also discuss the implementation of views in xTagger[2], a document-centric XML editor designed for work with multihierarchical markup (Section 3).

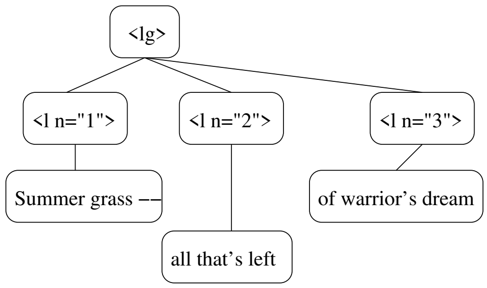
2. VIEWS FOR ELECTRONIC EDITIONS

Most software for editing XML documents uses, in general, two data structure representations: a document model for *data access* and a document model for *data presentation*. For *data access*, the XML Document Object Model (DOM) is the most common data structure used. It represents XML as a tree, provides API for its traversal and modification and supports path queries, transformations (XSL) and schema validation. The data model for the XML *presentation* varies from editor to editor and depends, in general, on the implementation of the editor. Unlike DOM, the data structure for document presentation is, typically, a list of elements. In this list each element represents a serialized form of an XML tag or text and contains style information for displaying the document. The presentation model also serves as the interface for document editing: text and markup updates are performed using API of the presentation model data structure. For example, let us consider the following XML document D:

```
<lg> <l n="1">Summer grass --</l>
     <l n="2">all that's left</l>
     <l n="3">of warriors' dreams.</l>  </lg>
```

The DOM tree of D [1] and a fragment of the document presentation data model for D are shown in Figure 1.

Let $dom(D)$ be the set of nodes in the DOM tree of D, and $list(D)$ be the set of elements in the data presentation model of D. We define a mapping $\mathcal{F} : list(D) \rightarrow dom(D)$ which returns the corresponding DOM node for each element in the presentation model. This mapping permits dynamic synchronization of the presentation and the data models: when an element $e \in list(D)$ is modified, the corresponding $\mathcal{F}(e) \in dom(D)$ is updated accord-



Element	Description	Style
<lg>	Start tag	color=red
<l n="1">	Start tag	color=red
Summer grass --	Text	color=black
</l>	End tag	color=red

Figure 1: The DOM tree and the document presentation data structure (excerpt).

[*]Work supported, in part, by the NEH grant RZ-20887-02.

[†]Work supported, in part, by the NSF grant ITR-0219924 and the NSF grant ITR-0325063.

[1]We label an element node with the corresponding tag name plus, where necessary, the attribute name and value pairs.

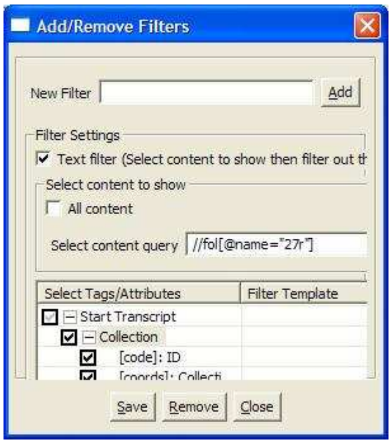


Figure 2: Defining views (filters) in xTagger.

ingly. The inverse mapping, $\mathcal{F}^{-1} : dom(D) \rightarrow 2^{list(D)}$[2], associates with each node in $dom(D)$ the corresponding element(s)[3] in $list(D)$. The mapping $\mathcal{F}^{-1}$ allows, for instance, to map path query results, performed over DOM, to elements and ranges in the presentation model (for instance, to highlight the results).

Definition. *An XML view of an XML document D is a set of nodes $\mathcal{V} \subseteq dom(D)$.*

Using a view $\mathcal{V}$ of D we can define a reduced presentation model of D, $\mathcal{F}^{-1}(\mathcal{V}) = list(\mathcal{V})$. XML views are a useful tool in customization of the editorial process. Human editors can define views of the document, that match their tasks at hand: e.g, a views showing all markup for a single chapter of the document, or a view, showing only the sentence markup for sentences which contain portions reconstructed by the editors. For a single-hierarchy XML document, XML views can be represented as XPath queries.

For multihierarchical XML documents, we have proposed to use a KyGODDAG data strcuture[1] in place of DOM. KyGODDAG[4] combines the DOM trees for each individual hierarchy in one structure with shared root and content. We have also extended XPath, to path expressions over KyGODDAG[1]. In addition to standard XPath functionality, our extension allows for path expressions to transcend hierarchy borders and query for such characteristic properties of multihierarchical XML as markup overlap between nodes from different hierarchies. Thus, XML views over multihierarchical XML documents can be represented as path expressions in extended XPath over KyGODDAG.

3. IMPLEMENTING VIEWS

In [2] we have described xTagger, the editing tool for document-centric multihierarchical XML. Here, we briefly describe how xTagger had been extended to handle XML views, in particular, definition of views, visualization of views, and views maintenance.

View definition. In xTagger we define views (a.k.a. *filters*) in two ways (Figure 2): (i) based on element names and, if needed, attribute names and values (possibly specified by regular expressions); (ii) based on a node set selected by an Extended XPath expression. In the former view definition all descendent text nodes of the selected element nodes are included in the presentation model. In the latter view definition, text nodes are explicitly included in the data presentation model if they are in the node set result of the path expression evaluation.

Visualization. Based on a defined view, $\mathcal{V}$, xTagger generates a document presentation, $list(\mathcal{V})$, and a mapping $\mathcal{F} : \mathcal{V} \rightarrow list(\mathcal{V})$. The XML document is then displayed based on the current data presentation $list(\mathcal{V})$.

View maintenance. Changes of document are made at the presentation model level, $list(\mathcal{D})$. In turn they are mapped to the document access model level, $\mathcal{D}$. The mapping $\mathcal{F}$ is updated accordingly. If $\mathcal{V}$ is affected but the change, $\mathcal{F}(\mathcal{V})$ is updated accordingly, and the change is reflected in the view display. It is possible that markup insertions or deletions create inconsistencies between data access model and data presentation model: $\mathcal{F}^{-1}(list(V)) \neq \mathcal{V}$ (that is, nodes not included in $list(\mathcal{V})$ or included when they are not supposed to be included). To prevent this situation, xTagger reevaluates the presentation model, $list(V)$, after each markup insertion or deletion.

4. CONCLUSIONS

In situations when human editors have to deal with a wide range of diverse markup in their work on annotating documents, XML views let them select for display only the features currently of interest and/or relevance. Extended XPath expressions allow users to express a wide range of views over multihierarchical XML documents. This functionality has been implemented in our document-centric XML editor tool, xTagger. Not surprisingly, our experience with implementing xTagger shows that the more presentation features an XML editor provides (such as different colors and text styles for tags, attribute names and values, entities, etc.) the more memory the presentation model uses. We have found that the memory used by the presentation model is by far greater than that used by the access model. In practice, our approach of XML views can greatly reduce the editor memory requirements while preserving full editing capabilities.

Our current work in this area concentrates on the problem of incremental maintenance of XML views described in this paper. When a document is updated, the answer set for an extended XPath query can change, sometimes significantly. We are in the process of developing efficient algorithms for incremental updates to XML views and for determination of situations when such updates are applicable.

5. REFERENCES

[1] I. E. Iacob and A. Dekhtyar. Towards a Query Language for Multihierarchical XML: Revisiting XPath. In *In Proc. of the International Workshop on the Web and Databases (WebDB)*, pages 49–54, 2005.

[2] I. E. Iacob and A. Dekhtyar. xTagger: a new approach to authoring document-centric XML. In *JCDL '05: Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, pages 44–45, New York, NY, USA, 2005. ACM Press.

[3] I. E. Iacob and A. Dekthtyar. Building Tools for Image-Based Electronic Editions. In *Proc., Joint Conference of the ALLC and ACH, Victoria, BC, Canada*, 2005.

[4] C. M. Sperberg-McQueen and C. Huitfeldt. GODDAG: A Data Structure for Overlapping Hierarchies. In *Principles of Digital Document Processing, DDEP/PODDP 2000, Munich*, pages 139–160, Sept. 2000. Early draft presented at the ACH-ALLC Conference in Charlottesville, June 1999.

[2] The $2^{list(D)}$ notation represents the set of all parts of the set $list(D)$.

[3] An element node in $dom(D)$ is associated to a start and an end tag in $list(D)$; a text node in $dom(D)$ is associated to a text element in $list(D)$.

[4] KyGODDAG is derived from a more generic GODDAG data structure [4].