Multihierarchical XQuery for document-centric XML

Ionut E. Iacob[*]
Department of Mathematics & Statistics
UNC Wilmington
Wilmington, NC, USA
iacobi@uncw.edu

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

ABSTRACT

Text has a non-hierarchical structure. Not surprisingly, searching for information in the content of a document often yields results that overlap the structure within the document. It is often of a great interest to relate such results to the embedded document structure.

In this work we present an extension of the XQuery language over multihierarchical document-centric XML documents. We illustrate the benefits of using multihierarchical XQuery for text-and-structure searches in document-centric XML documents More specifically, multihierarchical XQuery allows representing relationships between text search results and document structure even for cases where such search results overlap markup boundaries and even in cases when only one markup hierarchy is considered.

General Terms

Querying concurrent markup

Keywords

XML, XQuery, multiple hierarchies

1. INTRODUCTION

Originally designed as a language for document markup, XML has seen the number and variety of its uses go far beyond the original intent. Recent development of XML-related standards, such as XML Schema and XQuery [3] is driven more by the use of XML as an underlying representation for data management applications. With incorporation of XML into major database management systems, and with continued adoption of XML as the medium for business applications, the continued use of XML for document annotation often flies under the radar.

[*]Work supported, in part, by a Collaborative Research Award from NEH (RZ-20887-02) and the Andrew W. Mellon Foundation.

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

XIME-P 2006, 3rd International Workshop on XQuery Implementation, Experiences and Perspectives, June 30, Chicago, Illinois

At the same time, document-processing and document-annotation applications that rely on XML are gaining popularity. The readability of XML and the availability of free processing tools are just two of the reasons for this popularity.

Use of XML for document markup poses a number of unique challenges. Complex document annotations give rise of multihierarchical markup – use of markup from multiple schemas to annotate the same text[15, 9]. This, in turn, leads to problems with markup representation and storage.

Another set of problems is associated with querying document-centric XML. Unlike data-centric XML data, the content of document-centric XML documents has meaning and value even when all markup is removed (typically, XML is used to annotate existing texts). User information needs range from requests that need to be represented as markup queries over multiple domains, to text searches that completely ignore markup boundaries.

In this paper, we discuss the means of expressing and implementing some of the typical information needs related to document-centric (and sometimes, multihierarchical) XML. Our contribution is threefold: (i) we extend XQuery with search capabilities over multihierarchical documents; (ii) we present an XQuery enhancement for text search using regular expressions; (iii) we show that the extension of XQuery for multihierarchical documents has applications beyond multihierarchical markup: complex text searches over single-hierarchy XML documents may greatly benefit from XQuery extensions to reveal important relationships between the text search results and the document structure. Although our implementation of the XQuery extension relies on an appropriate data structure representation of multihierarchical markup, we emphasize that both the data structure and the extended XQuery are generalizations of the DOM data structure and XQuery language respectively. The XQuery extension we present can be straightforwardly implemented on top of DOM representation of XML documents while preserving the advantages of the extended XQuery.

The rest of the paper is organized as follows. In Section 2 we briefly present a few case studies that motivate this work. The data structure model for the multihierarchical XML and the appropriate path language to navigate this data structure are described in Section 3. In Section 4 we describe the extended XQuery for querying multihierarchical XML and we present the related work and conclusions in Section 5.

gesceaftum unawendendne singallice sibbe gecynde þa

<r><line>gesceaftum unawendendne sin</line><line>gallice sibbe gecynde þa</line></r>

<r><vline><w>gesceaftum</w><w>unawendendne</w> </vline><vline><w>singallice</w><w>sibbe</w><w> gecynde</w></vline><vline><w>þa</w></vline></r>

<r><res>gesceaftum una</res>wendendne s<res>in</res>
<res>gallice sibbe gecyn</res>de þa</r>

<r>gesceaftum una<dmg>w</dmg>endendne singallice sibbe gecyn<dmg>de þa</dmg></r>

Figure 1: An Old English manuscript and four encodings of different manuscript features.

2. XML QUERIES IN ELECTRONIC EDITIONS

The problem of multiple (concurrent) markup hierarchies often occurs in document encodings when one wants to encode a wide variety of features in a single XML document. A typical example is building image-based electronic editions of manuscripts[13]. Such electronic editions of historic documents and document collections are beginning to emerge as important new digital resources for humanities scholars and the general public. These editions can provide numerous scholars with anytime first-hand access to digital images and the content of unique and fragile material that is not otherwise widely available for study.

We illustrate the problem of overlapping hierarchies in document-centric XML in the following example. Figure 1 shows an image of a manuscript[1], the content of the manuscript fragment, and encodings of four groups of manuscript features: physical manuscript organization (`<line>`), document structure (`<vline>` – verse lines, `<w>` – words), editorial restorations of the manuscript content (`<res>`), and manuscript condition (`<dmg>` – damages). There are two main reasons for representing the encodings as different XML documents: (i) each encoding corresponds to a semantically independent markup hierarchy, dictated by the nature of the application and (ii) there is overlap in the scope of some markup (see, for instance, `<line>` and `<w>` markup) so all markup cannot be *naturally* represented as well-formed XML[2].

The problem of representing documents with multiple markup hierarchies has been addressed by us in [10, 9]. Section 3 summarizes our results. For the rest of this section we discuss a few querying scenarios that are likely to occur for such documents. Then, in Section 4, we present our solutions for the problems presented here.

[1]Cotton Otto A vi, King Alfred's translation of Boethius "Consolation of Philosophy", a 10th century Old English manuscript.

[2]In [6] we show that representing such markup using "hacks" in XML comes with a steep price at query processing time.

XQuery is an attractive language for the document encoding community because it combines in itself both the querying and the XML transformation capabilities. The latter feature is extremely important as document encoding community is actively using XSL transformations for document presentation. The three querying scenarios presented below appear in practice and involve both the need to search the documents and the need to present the search results appropriately.

I. Searching for nodes in a hierarchy and their relationships with nodes in other hierarchies. Searching for words is a common task in processing document encodings. However, just retrieving individual words without surrounding context is insufficient under many circumstances. It is often the case that the words in the search result need to be shown within their "surroundings", such as the manuscript lines where they appear. Alternatively, the search for words can be conducted based not only on their spelling but also on certain structural properties, such as, words which are damaged in the manuscript. Such information needs require not only querying the markup, but also presenting the results in a way that is immediately understandable to the scholars studying the manuscript. For example, consider the following two search questions:

1. Find and display lines containing the word `singallice`.

2. Find and display lines containing words that are totally or partially damaged and highlight such words.

The first query seems to be straightforward. However, there is a subtle problem about it, as shown in Figure 1. In this figure, the word `singallice` is split between two manuscript lines. The second query raises some problems also: `<w>` and `<dmg>` markup are in different hierarchies, and therefore a relationship between them needs to be determined.

II. Searching for (sub)strings of the text content of element nodes. Another common task in document encoding processing is searching for substrings. Here, the queries can be formulated in a variety of ways: search for words starting/ending with a given substring, search for words containing a certain substring, search for specified consecutive words in a phrase. In general, such queries for strings or substrings can be captured by a regular expression. However, the substring text can be split by XML markup in the encoding. As is the case with the previous types of queries, presentation of the search results may play an important role here. Consider the following query:

1. Find all words that contain the substring `unawe`, display such words and highlight the substring matching(s).

As opposed to the first set of queries in *I.*, this query appears to target a single document hierarchy: the one that contains `<w>` markup. However, while XQuery has capabilities of identifying such words, highlighting matchings is not a straightforward task[3].

[3]XSLT 2.0 [12] has the ability of text matching and iterating over matching/non-matching substrings.

III. Searching for (sub)strings of the text content of element nodes in a certain relationship with other element nodes. Finally, the last situation we consider is more general than the previous situation: in addition to highlighting the matched (sub)strings, we have to highlight *parts of the (sub)string based on how they are marked up in the encoding.*

For instance, we consider the following query:

1. Find all words that contain the substring `unawe`, display such words and highlight (display in bold) the substring and italicize (parts of) matched substrings that were restored (that is, covered by `<res>` markup).

It is clear that the query requires searching over multiple hierarchies as `<w>` and `<res>` are in different hierarchies.

Before we proceed to answer these queries (Section 4), we describe the data model we use for multihierarchical documents and the extension of XQuery we define on top of this data structure.

3. MULTIHIERARCHICAL XML

Multihierarchical markup has been subject of attention ever since the inception of SGML [14]. Up until recently, most of the work on dealing with it had been done by document processing experts [15][4]. However, almost all work on multihierarchical markup was limited to finding proper serialization syntax for representing multiple hierarchies in a single XML document. In [6] we have shown that this approach tends to lead to inefficient solutions.

Our work on multihierarchical markup addresses the problem of its management at the level of representation. In this section we briefly outline our work to date [5, 10, 9] on representation and querying multihierarchical XML documents.

Multihierarchical document-centric XML. A *Concurrent Markup Hierarchy (CMH)* is a collection $\langle D_1, \ldots, D_n \rangle$ of DTDs[5], and an XML element r, such that r, called the root of the hierarchy, is present in each DTD, no other XML elements are shared by different DTDs, and in each D_i all elements $x \neq r$ are reachable from r[5]. Concurrent markup hierarchies specify the markup that will be used to create multihierarchical documents.

A *multihierarchical XML document* d over a CMH H, is a collection of XML documents $d_1, \ldots, d_n$ and a string S, such that for all $0 \leq i \leq n$, d_i is an encoding of S using markup from the DTD D_i, with root r [5].

KyGODDAG. XML documents are parsed into a data structure called DOM tree, using DOM parsers. Applications use DOM API to access the parsed XML[4]. In [10, 9] we proposed a data structure called KyGODDAG[6] to be used in place of a DOM tree for storage and access to multihierarchical documents. Consider a multihierarchical XML document $d = \langle S, (d_1, \ldots, d_n) \rangle$. Let $S = l_1 \cdot l_2 \cdot \ldots \cdot l_s$ be a partition of S[7], into *leaves*, longest substrings such that no markup in any of the $d_1, \ldots d_n$ breaks any substring l_j (that is, markup appears only at the substring boundaries). Then, a KyGODDAG G_d is a directed acyclic graph $G_d = \langle N, E \rangle$, where $N = \cup_{i=1}^{n} nodes(d_i) \cup \{l_1, \ldots, l_s\}$, and the set of edges consists of the following. First, if $n_1, n_2 \in nodes(d_i)$, then $(n_1, n_2) \in E$ iff (n_1, n_2) is in the DOM tree of d_i. Second, if n is a text node in some d_i and l is a leaf node, then $(n, l) \in E$ iff $l \subseteq content(n)$.

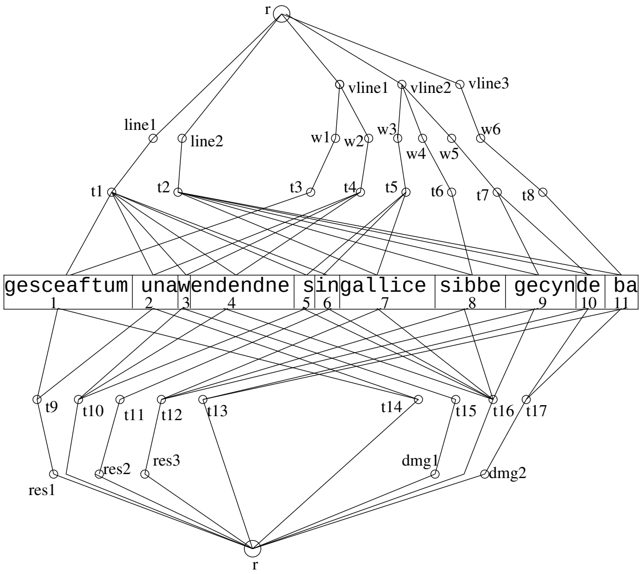

Figure 2: The KyGODDAG data structure for the encodings in Figure 1

Informally, a KyGODDAG for a multihierarchical XML document is a data structure that unites the DOM trees for all hierarchies at the root element, and creates a layer of *leaf* nodes that are connected to the text nodes from each hierarchy, which contain their content. The KyGODDAG data structure for the encodings in Figure 1 is represented in Figure 2 (for the sake of simplicity, two KyGODDAG roots were represented in the figure; they actually represent the same KyGODDAG node). In the figure, element nodes are labeled by their names followed by a number representing the order of the respective element node among nodes with the same name (for instance, `dmg1, dmg2`, etc.). The text nodes are labeled with `t1, t2`, etc. following the document order and the leaf nodes are represented by boxes labeled with numbers.

Path Expressions. In [9] we proposed a path expression language for multihierarchical XML documents and described its semantics over KyGODDAG. The language is an extension of XPath with a number of new axes for traversal of the KyGODDAG between different hierarchies, and a number of new node test that allow for simple access to content in individual hierarchies. All XPath axes are preserved in the extended language. XPath axes applied to non-root nodes return nodes within the same DOM tree component of the KyGODDAG. When applied to the root, they return nodes in all components. New axes can be broken into two

[4]See [5] for a survey of early research on multihierarchical markup.

[5]Or other XML schema descriptions.

[6]Sperberg-McQueen and Huitfeldt proposed a data structure called GODDAG, Generalized Ordered Descendant DAG, to be used for storage of multihierarchical document-centric markup[16]. Due to lack of a formal definition in [16], we we under impression that the structure proposed by us in [10, 9] was an instance of GODDAG. Personal communication with C.M. Sperberg-McQueen made us realize that it was, in fact, a different data structure, which we now call KyGODDAG.

[7]Here, $\cdot$ is the operation of string concatenation.

categories: multihierarchical versions of traditional XPath axes: xdescendant, xancestor, xfollowing, xpreceding, and new axes for representing markup overlap: preceding-overlapping, following-overlapping, overlapping.

To provide formal definitions of the new axes we need the following notation. Let $d, G_d = \langle N, E\rangle$ be a multihierarchical XML document and its KyGODDAG. Let x be a node in some hierarchy d_i of d. We let $ancestor(x)$ and $descendant(x)$ be respectively the ancestor and descendant nodes of x in its hierarchy. Then we let $leaves(x)$ denote the set of leaf nodes in the node set $descendant(x)$. Leaf nodes $l_1, \ldots, l_s$ of G_d come with a linear order: $l_j < l_k$ iff $j < k$. Given a set L of leaf nodes, $\max(L)$ and $\min(L)$ denote the maximum and the minimum leaf node in set L w.r.t. the above mentioned linear order. The semantics of the new axes is specified as follows [9].

DEFINITION 1 (EXTENDED PATH AXES). *We define the following extended path axes over KyGODDAG:*

- $xancestor(n) = \{m \in N|\ m \notin descendant(n) \cup \{n\}$ *and* $leaves(n) \subseteq leaves(m)\}$;
- $xdescendant(n) = \{m \in N|\ m \notin ancestor(n) \cup \{n\}$ *and* $leaves(n) \supseteq leaves(m)\}$;
- $xfollowing(n) = \{m \in N|\max(leaves(n)) < \min(leaves(m))\}$;
- $xpreceding(n) = \{m \in N|\min(leaves(n)) > \max(leaves(m))\}$;
- $preceding - overlapping(n) = \{m \in N|leaves(n) \cap leaves(m) \neq \emptyset$ *and* $\min(leaves(n)) \in (\min(leaves(m)), \max(leaves(m))]$ *and* $\max(leaves(n)) > \max(leaves(m))\}$;
- $following - overlapping(n) = \{m \in N|leaves(n) \cap leaves(m) \neq \emptyset$ *and* $\max(leaves(n)) \in [\min(leaves(m)), \max(leaves(m)))$ *and* $\min(leaves(n)) < \min(leaves(m))\}$;
- $overlapping(n) = following - overlapping(n) \cup preceding - overlapping(n)$.

The new node tests are described below(the *String* parameter is a comma-separated list of hierarchy names):

DEFINITION 2 (EXTENDED NODE TESTS). *We define the following node test extensions for path expressions over KyGODDAG:*

- *text(String): the node test is evaluated to true if and only if the context node is a text node in the specified hierarchy (hierarchies).*
- *node(String): the node test is evaluated to true if and only if the context node is any node type in the specified hierarchy (hierarchies).*
- **(String): the node test is evaluated to true if and only if the context node is a element node in the specified hierarchy (hierarchies)*
- *leaf(): the node test is evaluated to true if and only if the context node is of type* **leaf**.

KyGODDAG node order. Before we proceed to extending XQuery we need to extend the underlying data model. In particular, we need to define iterations over the extended axes evaluation (node sets from multiple hierarchies) and comparisons between nodes, possibly from different hierarchies. We define the order of nodes in a KyGODDAG in a way similar to XQuery's document order [3] as follows.

DEFINITION 3 (NODE ORDER IN KYGODDAG). *The relative order of KyGODDAG nodes is stable and subject to the following constrains:*

1. *the root node is the first*
2. *if two nodes are in the same hierarchy, then they follow the order in the respective hierarchy's DOM*
3. *if two nodes are in different hierarchies then their order corresponds to their hierarchies order (stable but implementation dependent).*

Implementation. The framework for management of multihierarchical document-centric XML document had been implemented as part of the Edition Production and Presentation Technology (EPPT)[8] for image-based electronic editions of text documents. It was presented in [7, 8].

4. XQUERY FOR MULTIHIERARCHICAL XML

In this section we show how to use the extended XQuery to answer the questions posed in Section 2. Due to space constraints and to the fact that some queries require HTML presentation of the search results, we skip the HTML wrapping around the XQuery query (in order to produce a valid HTML document). This would be a straightforward exercise, however.

For each of the following queries we assume that the default document is the KyGODDAG in Figure 2. That is, any path that starts with "/" is an absolute path starting at the KyGODDAG root. Moreover, for the rest of this section we consider that all functions are internal and, for simplicity, we drop the namespace "fn".

Before we get started with the queries, we give the definition of a new extended XQuery internal function, `analyze-string()`, which we use for text searching with regular expressions in a similar way to XSLT 2.0 [12].

DEFINITION 4. *We define*
fn:analyze-string(*$node as xs:node(), $pattern as xs:string) as xs:node()*
such that each of the following happens:

1. *a new KyGODDAG hierarchy is created and a hierarchy name is assigned to this hierarchy (say,* **res1**);
2. *the content of the input node* **$node** *is wrapped by a tag* **<res>** *in the hierarchy* **res1**;
3. *the regular expression* **$pattern** *is matched against the content of the input node* **$node** *and each matching string is tagged with* **<m>**, *also a node in hierachy* **res1** *and a descendant of* **<res>**;

[8]Available at: `http://beowulf.engl.uky.edu/~eft/`

4. *the regular expression **$pattern** may be given as a well-formed XML fragment (such as "**xxx<a>xxxxxx**"), in which case: (i) each start tag is replace by "(", each end tag is replaced by ")"; (ii) the resulted regular expression is matched against the content of **$node**, then each regular expression's group matching is tagged with the markup the group was originated from; (iii) all new markup is in **res1** hierarchy and descendant of **<res>** node;*

5. *all temporary hierarchies generated by an execution of this funtion are deleted after the entire query is evaluated.*

EXAMPLE 1. *For instance, let's consider that we apply the function **analyze-string()** on input node **<w>unawendendne</w>** and input pattern **.*un<a>awe.***.*
The evaluation will produce the following markup (which will be, temporary, part of the KyGODDAG):
<res><m>un<a>awe</m>ndendne</res>

The function defined above presents the advantage of matching text while matching groups can be clearly marked. Moreover, by creating a new "virtual" hierarchy, the search results can be exploited in the context of the existing hierarchy/hierarchies. Since the search results are likely to overlap existing markup boundaries, the use of the function even in the context of a single hierarchy makes sense if the extended XQuery axes are used (the virtual results hierarchy is likely to overlap the existent hierarchy).

In the following we present the extended XQuery queries for the searches formulated in Section 2.

I. Querying for nodes in a hierarchy and their relationships with nodes in other hierarchies

1. Find and display lines containing the word `singallice`.

```
for $l in /descendant::line
  [xdescendant::w[string(.) = 'singallice'] or
  overlapping::w[string(.) = 'singallice']]
return string($l)
```

Output:
```
gesceaftum unawendendne sin
gallice sibbe gecynde þa
```

2. Find and display lines containing words that are totally or partially damaged and highlight such words.

```
for $l in /descendant::line[xdescendant::w[
    xancestor::dmg or xdescendant::dmg or
    overlapping::dmg]]
return ( for $leaf in $l/descendant::leaf()
         return
           if ($leaf[ancestor::w and
                     ancestor::dmg]) then
              <b>{$leaf}</b>
           else $leaf
         , <br/> )
```

Output:
```
gesceaftum <b>una</b><b>w</b><b>endendne</b>
sin<br/>
gallice sibbe <b>gecyn</b><b>de</b>
<b>þa</b><br/>
```

II. Querying for (sub)strings of the text content of element nodes

1. Find all words that contains the substring `unawe`, display such words and highlight the substring matching(s).

```
for $w in /descendant::w
            [matches(string(.)),".*unawe.*"]
return (
 let $res := analyze-string(
             $w, ".*unawe.*")
 for $n in $res/child::*
 return if ($n/parent::m) then
           <b>{$t}</b>
         else $t
  , <br/> )
```

Output:

```
<b>unawe</b>ndendne<br/>
```

III. Querying for (sub)strings of the text content of element nodes in a certain relationship with other element nodes

1. Find all words that contains the substring `unawe`, display such words and highlight (bold) the substring matching(s) and italicize (parts of) matchings that were restored (that is, covered by `<res>` markup).

```
for $w in /descendant::w
            [matches(string(.)),".*unawe.*"]
return (
 let $res := analyze-string(
             $w, ".*unawe.*")
 for $leaf in $res/descendant::leaf()
 return if ($leaf/xancestor::m
             and $leaf/xancestor::res) then
           <i><b>{$t}</b></i>
         else if ($leaf/xancestor::m) then
           <b>{$t}</b>
         else $t
  , <br/> )
```

Output:

```
<i><b>unawe</b></i><b>ndendne</b><br/>
```

Implementation. We have implemented the extended XQuery with support for multihierarchical XML in Java. We used the grammar from [3], enhanced with support for extended XPath axes and extended node tests, and the Java Compiler Compiler[9] parser generator to generate a Java parser for XQuery expressions.

A side effect of using the function `analize-string()` is the appearance of a temporary markup hierarchy while the extended XQuery is processed. In our implementation, the document nodes introduced while processing an XQuery expression with `analize-string()` exist *only during XQuery expression evaluation*. Moreover, the output of such an XQuery expression evaluation is either a string or a *sequence of strings*.[10]

[9]JavaCC version 3.2, verb,https://javacc.dev.java.net/,
[10]This is also a consequence of the fact that we found this

5. RELATED WORK AND CONCLUSIONS

The importance of full-text querying for XML is clearly emphasized in [2] as well as in the W3C's working drafts for XQuery 1.0 and XPath 2.0 Full-Text [1]. Our work attempts to deal with situations where such full-text search results overlap document markup boundaries. Such situations are likely to appear when searching for text in document-centric XML documents. We propose a complete solution to determine relationships between such search results and document structure.

The work of Jagadish et al. [11] is a complete treatment of the multi-hierarchical *data-centric* XML problem: representation, processing, and querying. The authors highlight the importance of a proper query language for multiple XML hierarchies and leverage path expressions used in XQuery for searching XML. The XQuery language extension that Jagadish et al. propose increases the power of XQuery to take advantage of their Multi-Colored Trees (MCT) data structure for multiple data-centric XML hierarchies while preserving the original XQuery semantics when querying a single XML hierarchy. Unfortunately, MCTs do not support overlapping markup.

In our work of extending XQuery over concurrent markup hierarchies (with possible overlapping structures) represented as KyGODDAG, the following steps were taken: (i) we defined new axes and new semantics for accessing parts of XML with concurrent markup represented as a KyGODDAG [9]; (ii) we extended the XPath node test model to take into account multiple hierarchies and new node types of KyGODDAG; (iii) we defined a stable order over the nodes in KyGODDAG. Also, we presented an enhancement of XQuery for searching text using regular expressions and we argued that XQuery for overlapping hierarchies has potential applicability for searching text in XML documents and transforming XML data in other representations (XML, HTML, etc.). We have implemented the extended XQuery in Java and it serves as a main search and results presentation engine for the Edition Production and Presentation Technology (EPPT), a platform for preparing image-based electronic editions of manuscripts.

Our future work plans involve efficient implementation of extended XQuery over multihierarchical document structures and continuing our study of its applicability for full-text searches[1].

6. REFERENCES

[1] S. Amer-Yahia, C. Botev, S. Buxton, P. Case, J. Doerre, D. McBeath, M. Rys (Eds.), and J. Shanmugasundaram (inv.exp.). XQuery 1.0 and XPath 2.0 Full-Text. http://www.w3.org/TR/xquery-full-text/, 2005. W3C Working Draft November 2005.

[2] S. Amer-Yahia, L. V. S. Lakshmanan, and S. Pandit. FleXPath: flexible structure and full-text querying for XML. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 83–94. ACM Press, 2004.

[3] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Siméon (Eds.). XQuery 1.0: An XML Query Language. http://www.w3.org/TR/xquery/, Nov 2005. W3C Candidate Recommendation.

[4] M. Champion, S. Byrne, G. Nicol, and L. Wood (Eds.). Document Object Model (DOM) Level 1 Specification. http://www.w3.org/TR/REC-DOM-Level-1/, Oct 1998. World Wide Web Consortium Recommendation, REC-DOM-Level-1-19981001.

[5] A. Dekhtyar and I. E. Iacob. A Framework for Management of Concurrent XML Markup. *Data and Knowledge Engineering*, 52(2):185–215, 2005.

[6] A. Dekhtyar, I. E. Iacob, and S. Methuku. Searching Multi-Hierarchical XML Documents: the Case of Fragmentation. In *16th International Conference on Database and Expert Systems Applications (DEXA)*, volume 3588, pages 576 – 585, 2005.

[7] I. E. Iacob and A. Dekhtyar. A framework for processing complex document-centric XML with overlapping structures. In *SIGMOD '05: Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 897–899. ACM Press, 2005. Software demo.

[8] I. E. Iacob and A. Dekhtyar. Processing xml documents with overlapping hierarchies. In *Proceedings, Joint Conference on Digital Libraries (JCDL)*, page 409, June 2005. software demo.

[9] 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.

[10] I. E. Iacob, A. Dekhtyar, and K. Kaneko. Parsing Concurrent XML. In *Proceedings, 6th ACM International Workshop on Web Information and Data Management (WIDM 2004), Washington, DC.*, November 2004.

[11] H. V. Jagadish, L. V. S. Lakshmanan, M. Scannapieco, D. Srivastava, and N. Wiwatwattana. Colorful XML: one hierarchy isn't enough. In *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 251–262. ACM Press, 2004.

[12] M. Kay (Ed.). XSL Transformations (XSLT) Version 2.0. http://www.w3.org/TR/xslt20, Nov 2005. W3C Candidate Recommendation.

[13] K. Kiernan, J. Jaromczyk, A. Dekhtyar, D. Porter, K. Hawley, S. Bodapati, and I. Iacob. The ARCHway project: Architecture for research in computing for humanities through research, teaching, and learning. *Literary and Linguistic Computing*, 2004. forthcoming.

[14] A. Renear, E. Mylonas, and D. Durand. Refining our Notion of What Text Really Is: The Problem of Overlapping Hierarchies. *Research in Humanities Computing*, 1993. N. Ide and S. Hockey, (Eds.).

[15] C. M. Sperberg-McQueen and L. Burnard(Eds.). Guidelines for Text Encoding and Interchange (P4). http://www.tei-c.org/P4X/index.html, 2001. The TEI Consortium.

[16] 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.

extension useful in the context of formatting XQuery expression evaluation results in HTML.