META**XPath**

Curtis E. Dyreson
School of E. E. and C. S.
Washington State University, USA
cdyreson@eecs.wsu.edu

Michael H. Böhlen
Christian S. Jensen
Department of Computer Science
Aalborg University, Denmark
boehlen@cs.auc.dk csj@cs.auc.dk

Abstract

This paper presents the META*XPath data model and query language.* META*XPath extends XPath with support for XML metadata. XPath is a specification language for locations in an XML document. It serves as the basis for XML query languages like XSLT and the XML Query Algebra.*

The META*XPath data model is a nested XPath tree. Each level of metadata induces a new level of nesting. The data model separates metadata and data into different dataspaces, supports meta-metadata, and enables sharing of metadata common to a group of nodes without duplication. The* META*XPath query language has a level shift operator to shift a query from a data level to a metadata level.* META*XPath maximally reuses XPath hence the changes needed to support metadata are few.* META*XPath is fully compatible with XPath.*

Keywords: *Metadata, Query language, XML, XPath*

1 Introduction

The World-Wide Web ("web") is the world's most frequently used, text-based information resource. The web currently has several million servers providing access to several billion documents. Many of these documents conform to the HyperText Markup Language (HTML).

Metadata can be encoded in HTML using a `META` element. Metadata is literally "data about data." The metadata typically includes the the name of the author, the date of publication, and a description of the content of the document, especially with respect to a standard classification taxonomy. For example, the subject of an HTML document about a new strain of influenza could be lucidly described using the vocabulary of the US National Library of Medicine Medical Subject Headings (MeSH) [8] by prepending `META` elements with the appropriate Dublin Core [5] qualifiers for the subject.

In the near future, the Extensible Markup Language (XML) [12] is expected to replace HTML as the markup language of choice for web documents [2, 3]. XML is also expected to become an important language for web data exchange. XML is better suited to describing the structure and semantics of data because it is extensible. HTML has a pre-defined set of elements that, for the most part, describe the layout of a document. In XML, new elements can be invented to represent the semantics and structure of data.

Metadata can also be encoded in XML. Metadata in XML can have a complex structure and semantics, for instance the metadata might describe the type and schema of data to be exchanged. Proposals exist for relating XML data and metadata, cf. the Resource Description Framework (RDF) [6].

This paper presents a query language that combines XML data and metadata. There are many proposed query languages for XML [1, 7, 9, 10, 11, 13]. None of these proposals provide support for combining data with metadata. A data model and query language for metadata must address several concerns. First, metadata and data reside in different dataspaces. A single query should be able to combine constraints on data and metadata or query either independently. However a query on data alone should not *accidently* query metadata nor vice-versa. Second, metadata stands in relation to data as meta-metadata to metadata. Intrinsically a fact is neither data nor metadata, rather a fact is cataloged as data or metadata because of a relationship to other data. Third, some metadata has special semantics. Manipulations of data must faithfully observe these semantics. Fourth and finally, metadata not only describes but also proscribes data. For example security metadata is intended to restrict access to data.

The authors have previously described and implemented an SQL-like query language called AUCQL for a semistructured database that addresses all four issues raised above [4]. In this paper we extend XPath [10] with concepts borrowed from AUCQL. XPath is a specification language for locations in an XML document. It serves as the basis for XML query languages like XSLT [11] and the XML Query Algebra [13]. XML and semistructured data models are closely related [9], however there are extensive differences in system architectures between a database

server and a web server; hence in this paper we address only the first two issues raised above.

In Section 2 an example is given to motivate the utility of this research. Next the data model for METAXPath is given. The data model is a simple extension of the XPath data model in which metadata is represented by nesting document trees. Next, the query language for the extended data model is presented. Finally, the paper concludes with plans for future research. The presentation throughout the paper is informal. A reader interested in formal details should refer to the AUCQL paper [4].

2 Motivating Example

To exemplify our data model, consider the XML document for a person given below.

```
<?xml version="1.0">
<person ssn="234">
     <name>Ichiro</name>
</person>
```

Figure 1 shows the (logical) tree structure of the XPath data model for the XML fragment (including whitespace). Each node in the tree corresponds to a component of the document. A node is a list of properties (some properties have been omitted, e.g., text order). Each property is a datum about the node. The list varies depending on the *Type* of node. Only element nodes and the root node have children. The children of an element comprise the element's content. Note that in XPath the data model root is separate from the document root.

Assume that the following metadata (and meta-metadata) is available for the document.

- The URL at which the document resides is `www.wsu.edu/p.htm`.
- The `person` element is known to be in the English language. The following XML document describes the element.

  ```
  <?xml version="1.0">
  <language>English</language>
  ```

- The metadata about `person` being in English was authored by Suzuki as described in the following XML document.

  ```
  <?xml version="1.0">
  <author name="Suzuki"/>
  ```

The metadata given above is physically separate from the data, but to query the data and metadata together using XPath the metadata must be (logically) added to the tree structure. An inadequate strategy would be to add parent or child "`meta`" elements. This would be inadequate because the embedded metadata modifies the structure of the original document. An XSLT query to count the number of elements evaluated on the modified structure would return an incorrect result since it would count the metadata elements. The user could possibly rewrite the query to omit `meta` elements but users should not be forced to tinker with queries to contend with embedded metadata. Queries using the metadata should also be supported. For example it should be possible to retrieve elements that have metadata which is authored by Suzuki.

In the next section the XPath data model is extended to support metadata. In Section 4 we show how to extend the XPath query language for the new data model.

3 Data Model

Unfortunately the XPath recommendation [10] does not (currently) provide a formal data model; below we give an informal model that omits details extraneous to the aims of this paper.

A well-formed XML document is a collection of nested *elements*. An element begins with a start tag and ends with a paired end tag. Between the tags, an element might contain *content*, that is, text or other elements. The XPath data model is commonly assumed to be an ordered tree. The tree represents the nesting of elements within the document, with elements corresponding to nodes, and element content comprising the children for each node. Unlike a tree, the children for a node are *ordered* based on their physical position within the document.

The node types in the tree are element, processing instruction, comment, root, and text. Each node contains a list of properties. For example an element node has the following properties: *Value* (the element's name), *Type* (element), *Sibling order* (ordinal of text order among siblings), and *Attributes* (a set of name-value pairs, the attributes are unordered). Other properties, e.g., validation status, may also be present. In future, it may be possible to dynamically extend the information set (the XML Information Set proposal is available from the W3C but is undergoing extensive development).

3.1 The METAXPath data model

The METAXPath data model extends the XPath data model by adding an optional *Meta* property. The value of *Meta* is a METAXPath tree that represents the parsed metadata for that node. Note that metadata trees can be nested, that is, nodes in the metadata tree may themselves have *Meta* properties, which represent meta-metadata.

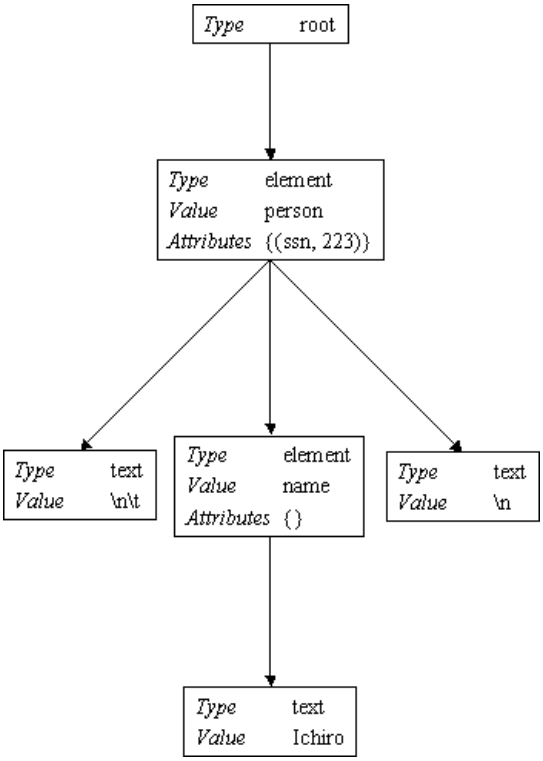

Figure 1. The XPath data model for the example fragment

Figure 2 shows the METAXPath tree for the example metadata given in Section 2. The URL metadata is in a tree that is hanging off of the data model root. The metadata tree's background is shaded gray in the figure. The language for the `person` element is a tree associated with the element. The language metadata tree is shaded with horizontal stripes. And the meta-metadata that Suzuki authored the language metadata is represented as the rightmost tree in the figure. The meta-metadata tree is shaded with diagonal stripes.

3.2 Sharing metadata in subtrees

To this point metadata has been associated with individual nodes. Often however metadata is common to a group of nodes. For example the author of a document is usually the author of each node in the document rather than just the document root. In this case the metadata should be shared among all the nodes in a subtree. In METAXPath two steps are needed to effect sharing. First, a *Meta* property must be added to each node to indicate the common metadata. Second, a metadata tree must inherit the metadata of all of its ancestors. This is accomplished by adding the children of the metadata trees for every ancestor to the metadata tree. Figure 3 shows the example METAXPath tree with metadata shared for every subtree. Each node has a *Meta* property that points to its metadata. Also the metadata for the `person` element must inherit the metadata of its ancestor(s). Hence the `source` element is a child of the root of the metadata tree for `person`.

3.3 Excluding shared metadata

In some situations it is essential to exclude metadata inherited from ancestors. For example assume that the `source` of the text `Ichiro` is `i.txt` rather than `p.htm`. In this case a new metadata tree must be created for the excluded node(s). Typically this involves creating a new root and adding children as needed. Figure 4 shows the example METAXPath tree with the URL metadata excluded for the `Ichiro` text node.

Sharing and excluding metadata increases the size of the METAXPath tree. Fortunately the sharing can be effected by pointers rather than copying nodes. Excluding data however will necessarily involve some copying. In the worst case each data node will have a unique metadata tree. We anticipate that sharing will be frequent and that exclusion for individual nodes will be rare.

4 Queries

In this section we extend the XPath query language to operate on METAXPath trees. Before the extensions are discussed we briefly summarize XPath.

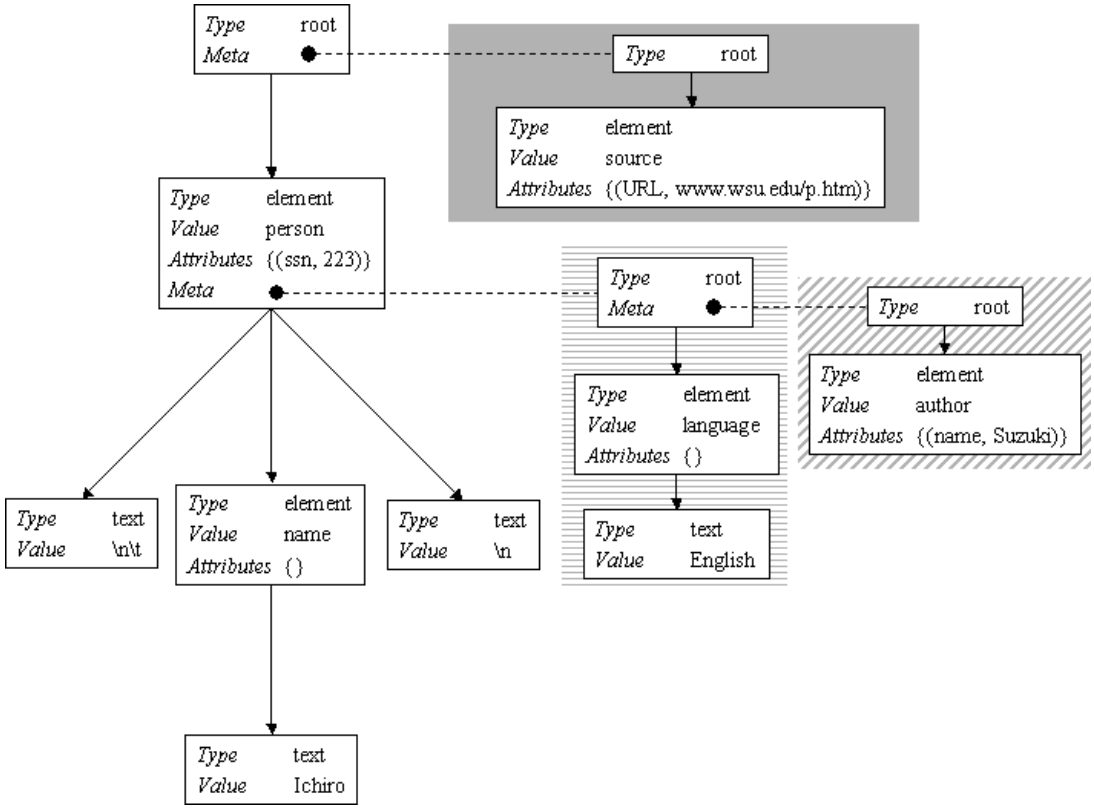

Figure 2. A METAXPath data model

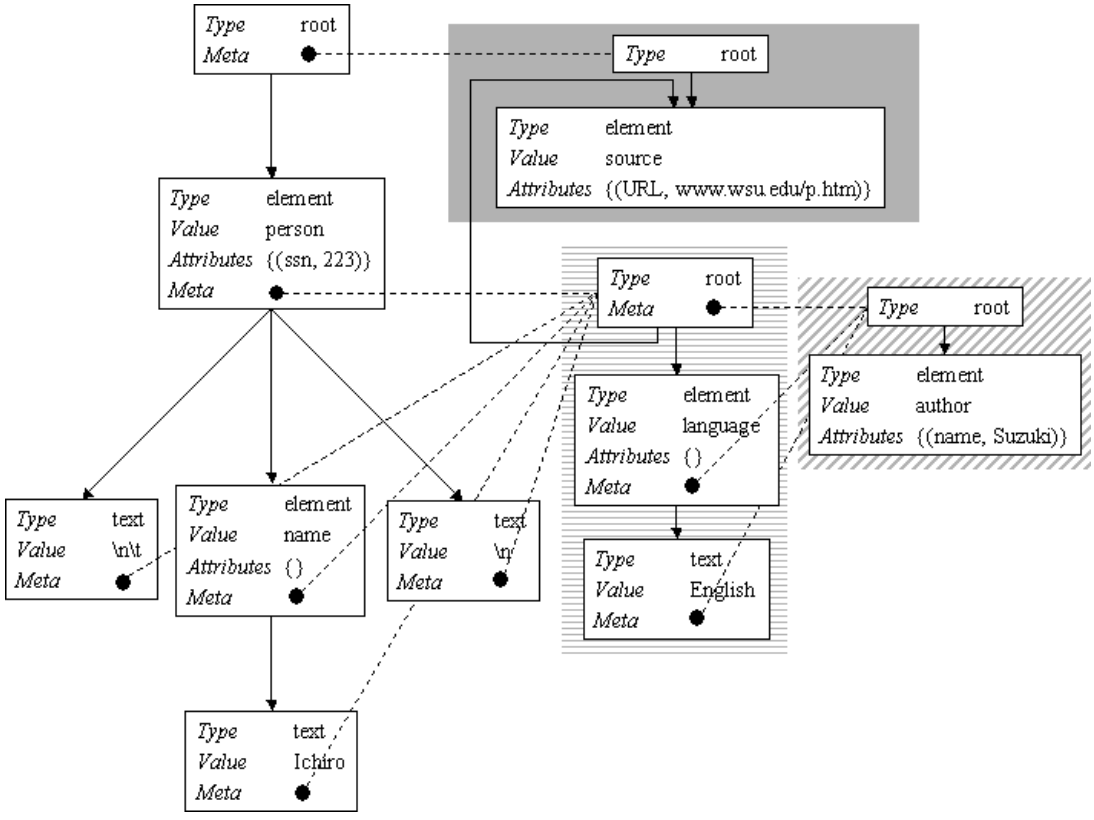

Figure 3. A METAXPath data model with shared metadata

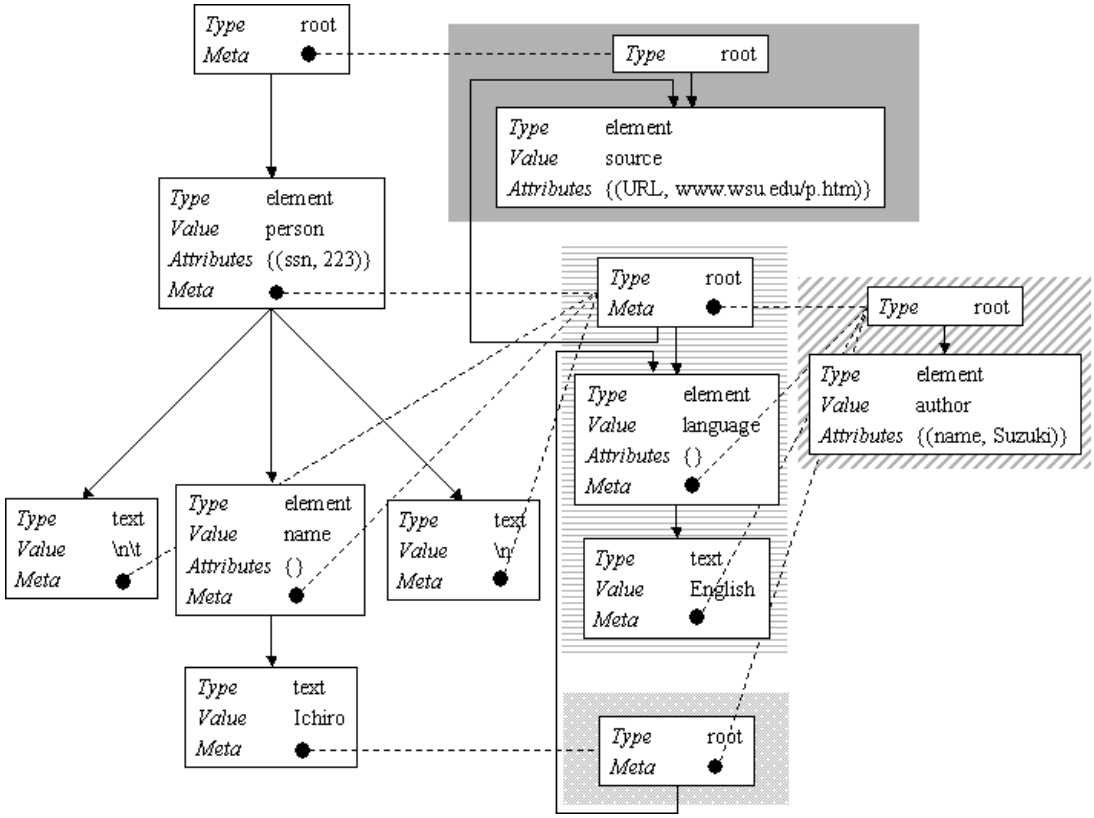

Figure 4. A META XPath data model with excluded metadata

4.1 XPath

An XPath query is a sequence of steps. Each step consists of four parts: a *context*, an *axis*, a *node test*, and a list of *predicates*. The *context* is the environment, including the context node, in which the step begins evaluation. The *axis* specifies a set of nodes, relative to the context node, that might be in the result of the step. Possible axes include `self`, `parent`, `child`, `descendent`, `ancestor`, `descendent-or-self`, etc. The *node test* is a predicate that is applied to each node in the axis. Possible node tests include `node()` and `comment()`. The node test is syntactically separated from the axis with the string '`::`'. Those nodes that pass the node test are then tested by the predicate(s). A step may have several predicates, each denoted by brackets. To qualify for a result, a node must pass every predicate. A predicate may itself include one or more XPath queries. A simple syntax for a step is given below.

$$axis\texttt{::}node\ test\texttt{[}predicate_1\texttt{]}\ldots\texttt{[}predicate_n\texttt{]}$$

The result of a step is an *ordered* list of nodes, called, paradoxically, a *node-set*. The ordering is based on the order in which the nodes appear in the document. The direction, *document order* or *reverse document order* relative to the context node, is determined by the axis (e.g., `child` is document order while `ancestor` is reverse). The result of a query is the result of the final step in the query. Nodes in the result of non-final steps are used (in order) as the context node for the next step. Syntactically, the steps are separated by the '`/`' character (or '`//`' or '`|`'). A simple syntax for a query is given below.

$$/step_1/\ldots/step_m$$

An example query to retrieve the children of the `course` element with a `code` attribute value of `CS451` is given below.

```
/descendent-or-self::course
 [attribute::code="CS451"]/child::*
```

The first step explores the `descendent-or-self` axis from the data model root. It applies an element test to keep only `course` elements. The predicate filters those nodes that lack a `code` attribute of `CS451`. The second step follows the `child` axis and retrieves any node (the wildcard is `*`).

XPath has an abbreviated syntax that shortens most queries. A shorter, semantically-equivalent query using the abbreviated syntax is given below.

```
//course[@code="CS451"]/*
```

Readers interested in further details should consult the XPath recommendation [10].

4.2 METAXPath

To extend XPath we make only a minor addition to the syntax and semantics. Observe that each nested tree in the METAXPath tree is a complete, well-formed XPath tree. This permits the unchanged use of XPath within each metadata level in the tree.

To XPath we add an operator to perform a "level shift." The level shift operator is a `meta` axis. The `meta` axis specifies the root node of the metadata tree (similar to how an `attribute` axis retrieves the value of the attribute property for a node). The axis supports only two node tests, both of which evaluate to true for the metadata tree root node: the wildcard ('*') and `node()`. Effectively the level shift just locates the metadata root node. The level shift is always "upwards" from the data to the metadata.

An example is given below. Assume that we want to locate nodes that are available from the URL `p.htm`. The following METAXPath query includes a level shift in the predicate.

```
/descendent-or-self::*
  [meta::*/child::source[
    attribute::URL="p.htm"]]
```

The query first explores the descendent axis below the data model root in the data tree. For each descendent it evaluates the predicate. In the predicate, the level shift moves the query to the metadata tree. The rest of the query determines if there is a `source` element child of the data model root (in the metadata tree) with the appropriate `URL` attribute. Note that the result of this query is a set of data nodes.

In abbreviated syntax, we will use a '^' to denote a level shift. It can appear anywhere that a '/' appears. The same query given above is given below using abbreviated syntax.

```
//[^source[@URL="p.htm"]]
```

4.3 Retrieving metadata

The query given above retrieves data nodes based on a metadata predicate. To retrieve metadata nodes the level shift operator can be used outside of a predicate. The following query retrieves `source` metadata nodes.

```
//^source
```

4.4 Querying meta-metadata

Level shifts are nested to query meta-metadata. The following query retrieves all nodes that have some metadata authored by Suzuki.

```
//[^^author[@name="Suzuki"]]
```

The query explores all data nodes. The first level shift explores to the root of the metadata tree for each data node. The second level shift then checks meta-metadata nodes for those that were authored by Suzuki. The following query retrieves all metadata nodes that have some metadata authored by Suzuki.

```
//^[^author[@name="Suzuki"]]
```

4.5 XPath compatibility

One very important point is that METAXPath is fully backwards-compatible with XPath. An XPath query on a METAXPath data model will simply ignore the metadata, and in fact cannot access the metadata under any circumstances. Since the METAXPath data model is completely backwards-compatible with the XPath data model all existing XPath queries on XML documents will continue to work when the document is queried using a METAXPath model.

5 Conclusion and Future Work

This paper briefly presents METAXPath, a data model and query language for XML data and metadata. The METAXPath data model is a nested XPath tree. Each level of metadata induces a new level of nesting. Metadata common to a group of nodes can be shared without duplication. The METAXPath query language has a level shift operator to shift a query from a data level to a metadata level. Since METAXPath maximally reuses XPath the changes needed to support metadata are few and METAXPath is fully compatible with XPath.

There remains much to do. XPath is only a part of query languages like XSLT. We need to investigate applying METAXPath techniques to a more complete query language. In particular we need to articulate operators for restructuring, grouping, and aggregating nested, metadata trees. We plan to implement METAXPath and empirically test the efficiency of our design. (For AUCQL we analytically showed that the space and time overhead on queries was linear in the size of the metadata.) It will be important to determine whether our suggested strategy for sharing and excluding metadata by copying pointers is efficient and maintainable in practice. Finally, although XPath lacks a formal semantics, we need to develop a formal semantics for METAXPath and reason about the completeness of the level shift operator.

References

[1] S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener. The Lorel Query Language for Semistructured Data. *International Journal of Digital Libraries*, 1(1):68–88, 1997.

[2] T. Berners-Lee. Keynote Address. In *Seventh International World Wide Web Conference (WWW7)*, Brisbane, Australia, April 1998.

[3] D. Connolly, R. Khare, and A. Rifkin. The Evolution of Web Documents: The Ascent of XML. *XML special issue of the World Wide Web Journal*, 2(4):119–128, Autumn 1997.

[4] C. Dyreson, M. Böhlen, and C. S. Jensen. Capturing and Querying Multiple Aspects of Semistructured Data. In *Proceedings of the International Conference on Very Large Databases (VLDB '98)*, pages 290–301, Edinburgh, Scotland, September 1999. http://www.eecs.wsu.edu/ cdyreson/AUCQL/.

[5] D. Hillmann. Using Dublin Core. http://dublincore.org/documents/2001/04/12/usageguide/, April 2001.

[6] O. Lassila and R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Technical report, W3C Technical Report, January 1999.

[7] B. Ludäscher, R. Himmeröder, G. Lausen, W. May, and C. Schlepphorst. Managing Semistructured Datat with FLORID: A Deductive Object-Oriented Perspective. *to appear in Information Systems*, 1998.

[8] NIH. Medical Subject Headings - Home Page. http://www.nlm.nih.gov/mesh/meshhome.html, June 2001.

[9] D. Suciu. Semistructured Data and XML. In *to appear in Proceedings of the International Conference on the Foundations of Data Organization (FODO '98)*, 1998.

[10] W3C. XML Path Language (XPath) Version 1.0. http://www.w3.org/TR/xpath, Nov. 1999.

[11] W3C. XSL Transformations (XSLT) Version 1.0. http://www.w3.org/TR/xslt, Nov. 1999.

[12] W3C. Extensible Markup Language (XML) 1.0 (Second Edition). http://www.w3.org/TR/REC-xml, Oct. 2000.

[13] W3C. The XML Query Algebra. http://www.w3.org/TR/query-algebra, Feb. 2001.