Sanitizing using Metadata in MetaXQuery

Hao Jin and Curtis Dyreson
Washington State University
Pullman, Washington
U.S.A.
{hjin, cdyreson}@eecs.wsu.edu

ABSTRACT
Metadata plays an important role in describing and proscribing data in both traditional and XML applications. In this paper, we present an extension of the XML data model and XQuery query language to certify the reachability of data and to sanitize data with the existence of metadata, especially prescriptive metadata. The data model extension is called MetaDOM, and the query language is called MetaXQuery. This paper describes a certify function to check if the metadata in the data model is correctly embedded, in other words, whether all of the data nodes are reachable from the root. It also describes a sanitize function that automatically corrects the data model if it is invalid. The sanitize function can also be used to generate a view of the data from a specific metadata perspective.

Categories and Subject Descriptors
H.2.1 [Database Management]: Logical Design – Data Models; and D.3.1 [Programming Languages]: Formal Definition and Theory – Semantics, Syntax.

General Terms
Algorithms, Management, Languages.

Keywords
Metadata, XML, XQuery, Data reachability.

1. INTRODUCTION
This paper outlines an extension to the XML data model to support metadata, especially prescriptive metadata and XQuery [20] to certify the reachability of data and sanitize data with metadata. Metadata is “data about data”. Metadata is usually descriptive, but some metadata, especially security metadata, is also prescriptive in the sense that it restricts access to data. We make two extensions to XQuery. The first extension is to the data model, which we call MetaDOM. Since MetaDOM is similar to both a previously described semi-structured model [7] and MetaXPath data model (lacking prescriptive metadata) [8], we only briefly summarize MetaDOM in this paper. The second extension is a collection of XQuery functions to support certifying and sanitizing data with metadata. The certify function checks the reachability of every node in an XML data model for correct embedding of metadata. The sanitize function can also be used to generate a view of the data based on a given metadata condition. These functions are the primary contribution of this paper.

This paper is organized as follows. The next section presents an example that motivates this research. We then summarize MetaDOM. Section 4 presents the XQuery extensions. Finally, we present related work and conclusions.

2. MOTIVATING EXAMPLE
Our motivating example comes from the W3C XQuery and XPath Full-Text Use Cases document [22]. The example scenario is a collection of book information that has both data and metadata. The metadata includes a book’s title, list of authors, publication information, and subject in several different languages. The data is the content of the book, which includes the full text of its chapters. We adopt a unified data model that supports prescriptive metadata, is able to distinguish data from metadata, and yet also capture their relationship. Suppose that a publisher makes information about its books available only to on-line subscribers that pay for the service. The publisher annotates the book data with security metadata. The security is intended to limit access to only the user Joe. The publisher also wants to archive the book information and so decides to record the transaction time of the book data. The transaction time is the system time when the data is available to the users. An example of such a metadata-aware data model is shown in Figure 1. The metadata and data in the model are separated into different, yet related scopes. The key difference between our MetaDOM model and a normal XML data model is the directed edge from book to metadata. These directed edges represent the relationship between data and its metadata. By following an edge, a user jumps to a metadata scope. The scope limits the search performed by a “wild card”. For instance a query that follows the descendant axis from the book node will not descend into the metadata.
Now let’s take a look at the challenges that we have with this metadata-aware data model. The first challenge is how to guarantee the reachability of each node with the embedded metadata in the data model. In a traditional well-formed XML document, every node is reachable from the root of the document. But *proscriptive* metadata can impose constraints on the data and affect the validity of the path descending from the root to a node. For example, if a parent node exists in the document in (transaction) time 2 to 5, then all its children and descendant nodes should only exist within that time period too, since otherwise, a child cannot exist independent of its parent in a well-formed XML data model. If one of the children exists in a completely different period, there is something wrong with either the parent or the child. Similarly, if a parent node is only accessible to the user Joe, then all its children and descendants should be only accessible to the user Joe too. Sometimes the metadata in a document could be wrong due to a number of factors, so we need to be able to check if the metadata is correctly embedded in the document and that every node in the data model is reachable, under some combination of proscriptive metadata. Moreover, if there’s something wrong with the metadata in the document, then it should be possible to identify and, in some cases, clean up or rectify the inconsistency.

The second challenge is how to apply the implicit semantics of metadata in user queries. For example, if a particular user, say Joe, queries the data model, then Joe should only see the part of document to which he has access. Or if the user wants to query the current version of the data, he should only see the data that has a transaction time including now. Some queries involve explicit metadata conditions (e.g., find book nodes from the perspective of user Joe and the current transaction time). The data model has to implicitly enforce security restrictions and the semantics of transaction time to correctly answer such a query. The process of *sanitization* enforces a specific metadata perspective throughout a data model. Sanitization will construct a new data model limited to the selected metadata perspective, which is similar to generating a (materialized) view of the data based on a set of metadata conditions. A user can subsequently query the view without explicitly giving these metadata conditions. For instance, if user Joe wants to find out the books in Figure 1 that are currently available, he could sanitize the `books` element with a security of Joe and a transaction time of `now`. The leftmost `book` node in Figure 1 will be discarded in the result because its transaction time metadata does not include the current time (now). The result is shown in Figure 2. In summary, sanitizing the data is the natural application of implicit metadata conditions.

![Figure 1 A part of the MetaDOM for the online publisher](image1)

![Figure 2 Books that are available to user Joe now](image2)

### 3. METADOM

MetaDOM extends the XQuery 1.0 and XPath 2.0 data model [21] by adding an optional *meta* property to a node’s Information Set. But in all other respects, the data model is the same as the existing model. The value of the *meta* property is a reference to the root node of a nested MetaDOM, which describes the metadata for this node. The metadata can be recursively nested, that is, a node in MetaDOM may itself have a *meta* property. Each level of nesting adds another level of metadata. Below is a formal definition of the MetaDOM node constructor. We adopt the notation used by the W3C (e.g., in [20]) to express all XQuery and DOM extensions used in this paper.

**Definition** [node constructor] Each node in MetaDOM is an extension of a DOM node. It has all the properties in the node’s information set, and adds a meta property. A MetaDOM node is cloned from a DOM node using the following constructor. The parameters are the node name, the sequence of its children, the metadata for the node, and the source from which all the other items in the information set are copied to the new node.
To access the metadata, we define an accessor function for the \textit{meta} property.

\textbf{Definition} [meta accessor] The meta accessor returns either a meta element node, which is the root of the metadata fragment that’s associated with the input node if it exists, or an empty sequence if there’s no metadata describing the input node.

\begin{verbatim}
meta-dm:node($name as xs:string,
  $children as node()*,
  $metadata as meta-dm:node(),
  $data as node()) as meta-dm:node()
\end{verbatim}

4. \textsc{METAQUERY}

This section presents an operation matrix that allows users to flexibly define the semantics of each metadata property as well as extensions to XQuery to support certifying the reachability of data and sanitizing data with metadata. The extensions include adding a \textit{meta} axis in XPath expressions, and a set of functions for certifying and sanitizing. These functions can be implemented by extending an XQuery processor, or in a programming language like Java and loaded into XQuery as \textit{external} functions. Parts can also be written directly in XQuery as \textit{user} functions. We will leave it to the MetaXQuery implementation to decide how to achieve these functions.

4.1 Operation Matrix

Since different metadata properties can have different semantics and constraints on the data, and introducing new metadata properties can add new semantics to the application, our framework uses an operation matrix that allows a metadata management system to tailor the semantics of each kind of metadata. There are several typical types of metadata.

\begin{description}
\item[Time.] The temporal metadata involves two dimensions, valid time, which is the real world time of a datum, and transaction time, which is the time it is stored in the system. For our purposes, their semantics are similar so we will treat them as a generic time dimension. The time will be a set of intervals (a temporal element [13]).
\item[Security.] The security metadata that we will use in this paper is a list of the users that are allowed to access the corresponding data and its descendants. Users who are not listed can’t access the data.
\item[Reliability.] Reliability is a measurement of the quality of the data. It could be a ranking of its quality by experts or users, or trustworthiness of the information source. We will use it as a numeric scale that ranges from 0 to 1.
\item[Language.] Language is a description of the natural language in which the data is written, e.g., English.
\end{description}

Table 1 lists the data type for each kind of metadata. The meta-value function retrieves a value of the specified return type for the indicated kind of metadata.

Table 2 is one possible operation matrix for the metadata properties used in this paper. The matrix is specific to a metadata application, so other semantics can be defined as needed. New metadata properties can also be defined as needed. In MetaXQuery functions, the matrix will be represented as a hash table, denoted \texttt{meta-dt:MetaENHash}. The hash table maps the name of the metadata property (e.g., time, security) and the desired operation to the metadata-specific operation. For instance, when sanitizing time metadata in the evaluation of a MetaXQuery expression, the application would call the sanitize-intersection function. Rather than defining each operation now, we will describe and define each operation when it is used in this paper.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|}
\hline
\textbf{Property} & \textbf{Certifying} & \textbf{Sanitizing} \\
\hline
\texttt{Time} & \texttt{certify-subset} & \texttt{sanitize-intersection} \\
\hline
\texttt{Security} & \texttt{certify-subset} & \texttt{sanitize-intersection} \\
\hline
\texttt{Reliability} & true & No-Op \\
\hline
\texttt{Language} & true & No-Op \\
\hline
\end{tabular}
\caption{The operation matrix for metadata properties}
\end{table}

4.2 MetaXPath

In order to make the meta accessor available to users of MetaXQuery we adopt the same approach used in MetaXPath [8]: we add a new axis, the \textit{meta} axis, to XPath expressions. The \textit{meta} axis follows the \textit{meta} property of the context node to its metadata. We will use the notation “\texttt{meta::}” to denote the meta axis. For example, “\texttt{meta::node()}” will locate the metadata’s root node from the context node. The meta axis will be abbreviated as “\texttt{~}”, In the data model in Figure 1, the MetaXPath expression “/book/transaction_time” will return the transaction_time element in the metadata of the book element.

The meta axis is orthogonal to the other XPath axes. In other words, the metadata can only be located through the meta axis. This ensures the separation of scope between data and metadata. So if we want to ask a query such as “When was book 1 in the document?”, it can be constructed in MetaXPath as follows.

\begin{verbatim}
/book[@number=“1”]/transaction_time
\end{verbatim}

Note that traditional XPath expressions can still be evaluated in MetaDOM. The query will just ignore the metadata. So it’s fully backwards-compatible to current XPath and XQuery.

4.3 Certifying Data Reachability

As mentioned in the first challenge of our motivation, we have to deal with the reachability of data with regard to metadata conditions, and that’s the certify function does. The certify function checks the consistency of the metadata between parent and child nodes in a data model, or in other words, whether a child can be reached from a parent. Given a parent node and its metadata, the metadata of each child must be consistent with the metadata of the parent in order for it to be reachable. The consistency check for each kind of metadata is defined in the operation matrix. If the check succeeds, the metadata is consistent.
Certify starts from the input node and first checks the metadata relationships between this node and all its children. After all the children have been checked, it recursively calls itself on each child to check the next level down the tree. If anywhere in the tree, an incompatible metadata value is found, the function will immediately return false, which means some descendant of the input node is unreachable. The descendent can be found and removed by sanitizing the data model as described in the next section. There may be several levels of metadata (e.g., meta-metadata), so certify checks each level of metadata for consistency.

**Definition** [metadata-specific certifying functions] These functions are used to certify specific kinds of metadata. The certify-subset function makes sure the metadata value of the child is a subset of its parent’s.

\[
\text{meta-fn:certify-subset} = \begin{cases} 
\text{True} & \text{If meta-fn:meta-value($\text{cmNode} / *$) } \subseteq \text{meta-fn:meta-value($\text{pNode} / *$)} \\
\text{False} & \text{Otherwise} 
\end{cases}
\]

### 4.4 Sanitizing a Data Model

If the certification of a document/fragment fails, we then have to correct the metadata to make it valid. Rather than relying on manual correction, we introduce a sanitize function that automatically constructs a certified data model. As mentioned in the second challenge of the motivating example, the sanitize function could also be used to generate a view of the data based on certain metadata conditions. This can be effected by sanitizing with respect to a selected metadata perspective. So for queries like “Which books are available to user Joe online?”, what we could do is first assign user Joe as the only metadata security to the root node and sanitize the entire document. Then a simple XPath query like ‘//book’ on the sanitized data model will locate books that are available to the user Joe.

The sanitize function starts by checking the metadata of each of the input node’s child and the metadata fragment of the input node as well. It then recursively calls itself on each of the existing child after the first step. If the processing of a certain type of the child node’s metadata returns NULL, this child node is unreachable from its parent and therefore will be removed from the result fragment.

**Definition** [sanitize] The sanitize function takes an input node, which is the root of the document/fragment and returns the root of the new valid document/fragment.

\[
\text{meta-fn:sanitize} = \begin{cases} 
\text{meta-dm:node(dm:node-name($\text{pNode} / *$), $\text{pmNode} / *$), $\text{cmNode} / *$) } & \text{if meta-fn:meta-value($\text{pNode} / *$) } \subseteq \text{meta-fn:meta-value($\text{pmNode} / *$)} \\
\text{null} & \text{Otherwise} 
\end{cases}
\]

Internally, the sanitize function calls the meta-sanitize function, which modifies the metadata of a child node to be compatible with its parent’s.

**Definition** [meta-sanitize] The meta-sanitize function takes an operation matrix and a pair of meta element nodes from a parent and a child node, and creates a new meta element node for the child. The new node represents a metadata fragment in which all its values are compatible with the parent node’s metadata.

\[
\text{meta-fn:meta-sanitize} = \begin{cases} 
\text{meta-dm:node(dm:node-name($\text{cmNode} / *$), $\text{pmNode} / *$), $\text{cmNode} / *$) } & \text{if meta-fn:meta-value($\text{cmNode} / *$) } \subseteq \text{meta-fn:meta-value($\text{pmNode} / *$)} \\
\text{null} & \text{Otherwise} 
\end{cases}
\]

**Definition** [metadata-specific sanitize functions] These functions are used to sanitize specific kinds of metadata. The sanitize-intersection function calculates the intersection of two metadata values of the same type and creates a new fragment representing that value.

\[
\text{meta-fn:sanitize-intersection} = \begin{cases} 
\text{meta-dm:node(dm:node-name($\text{cmNode} / *$), $\text{pmNode} / *$), $\text{cmNode} / *$) } & \text{if meta-fn:meta-value($\text{cmNode} / *$) } \subseteq \text{meta-fn:meta-value($\text{pmNode} / *$)} \\
\text{null} & \text{Otherwise} 
\end{cases}
\]

### 5. RELATED WORK

RDF is perhaps the most widely used language for annotating a document with metadata [19]. It has also become an important language for supporting the Semantic Web [1]. Several strategies for unifying the representation of XML and RDF have been proposed [12][17], but query languages have largely targeted either RDF or XML. There have been several RDF query languages proposed in the literature such as RQL [16], TRIPLE [18] and etc. For a comparison of these RDF query languages, please refer to [11]. Discussions are ongoing about accessing and
querying RDF data [19] but in general RDF query languages use a very different data model than the family of XML query languages. Support for particular kinds of metadata has been researched. Two of the most important and most widely discussed types of (proscriptive) metadata are temporal metadata and security metadata. Temporal extensions of almost every W3C recommendation exist, for instance, TTXPath [5], tXQuery [9], and tXSchema [3]. Grandi has an excellent bibliography of time-related web papers [10]. Security in XML management systems has also been researched, e.g., [2] and [4]. Our approach is to build infrastructure that supports a wide range of different kinds of metadata in the same vein as our previous efforts with the semistructured data model [7] and XPath data model [8].

6. CONCLUSIONS
In this paper, we outline an XML data model, called MetaDOM, which supports data annotated with metadata. Different semantics can be given to different kinds of metadata using MetaDOM. We show how to extend DOM to implement MetaDOM. We also present part of a query language, called MetaXQuery, for the extended data model. We focus on the reachability and sanitizing aspects of MetaXQuery. MetaXQuery extends XPath with an additional "meta" axis, and extends XQuery with additional functions. The certifire function checks the metadata relationship for reachability of each node in the data model. The sanitize function cleans the data model to guarantee every node in reachable, and it can also be used to generate a view of the data with a given metadata condition. These functions provide rich expressive power to our query language.

7. REFERENCES