A Data Model and an XQuery Extension for Concurrent XML Structures

Emmanuel Bruno and Elisabeth Murisasco
LSIS - UMR CNRS 6168
Université du Sud Toulon-Var, BP 20132
83957 La Garde Cedex, France
E-mail: {bruno,murisasco}@univ-tln.fr

Keywords: multistructure, concurrent hierarchy, textual document, tree-like structure, XML data model, XQuery

Received: March 13, 2009

An XML document is mainly hierarchical, but some applications need to simultaneously associate more than one hierarchy to the same data. In general, concurrent hierarchies cannot be merged in order to get a well-formed XML document. This work stands in this context: it aims at describing and querying hierarchical XML structures defined over the same textual data in a concurrent way. Our proposal called MSXD is composed of a data model (which can be seen as an index over the data and between all the structures) and a dedicated query language defined as an extension of XQuery. The key idea is to propose a method for a compact description of multiple tree-like structures over a single text based on segmentation. Segmentation encoding allows querying overlap/containment relations of markups belonging to different structures. This paper also tackles a solution to define a multistructured schema in order to describe the relationships (as weak constraints) between parts of concurrent structures. Finally, this paper focuses on the architecture of the XML environment implementing our proposals.

1 Motivation

XML [10] is the de facto standard to describe structured data. Several applications in the context of information systems are based on their use: electronic publishing, language engineering, technical documentation, digital libraries, Web, etc.

XML documents are mainly hierarchical. The hierarchy, captured in a tree-like structure [23], corresponds to one level of analysis of the data contained in the document (e.g., a logical analysis). A large set of tools are now available and widely used in particular for edition, querying (XPath [16], XQuery [7]) or transformation (XSLT [15]). According to us, this success is due to the hierarchical structure which is easier to exploit compared to the graph structure. Moreover, manipulation remains simple in an XML environment.

Recently, the title of an article published in the SIGMOD conference [27] claims that “One hierarchy is not enough” for data-centric application context. Indeed, the CONCUR feature of SGML [24] first pointed out this need in the nineties but in context of document-centric encoding where some applications need to consider more than one hierarchy over the same text in a concurrent way. These last years, several other works about concurrent markups have been published [31, 30, 32, 22, 33, 19, 25, 14, 21]. All these different works propose solutions to describe multiple tree-like structures (with overlapping) into a single document. The main problem with concurrent hierarchies is that they cannot be merged in order to get a well-formed XML document without using a flat representation or hyperlinks that make the structure difficult to query.

Our work stands in this context. It aims at representing and querying concurrent hierarchical XML structures defined over the same textual data. We call such data “multistructured textual documents”. The key idea is to propose a method for a compact description of multiple trees over a single text based on its segmentation. Segmentation encoding allows querying overlap/containment relations of markups belonging to different structures. The solution proposed, called MSXD for MultiStructured XML Documents\(^1\), is entirely XML compatible.

This paper describes on the first hand the MSXD model and its relative query language, and on the other hand, it details the way these proposals are implemented under an XML environment. We also tackle the description of relationships between structures (as weak constraints) in order to propose a solution to define a multistructured schema enabling validation across multiple hierarchies [20].

This paper is an extended version of different preliminary contributions ([11], [12] and [13]).

1.1 A running example

To illustrate the problem, we consider a medieval manuscript (Princeton, Garrett 80 (PG)) related to medico-pharmaceutical recipes written in Occitan language [8]. This kind of text is studied by philologists of the University

\(^1\) Funded by the french research agency (ANR) – Semweb project (2004-2007)
of Pisa (Italy) in the Department of Language and Romance Literature. Philology is a science that studies ancient or medieval civilizations by the mean of literary documents.

Researchers are interested in studying the text of the manuscript according to different points of view corresponding to different uses of the document. Each analysis results in a mainly hierarchical markup of the text which could be easily represented in XML.

From the document represented as an image (see Figure 1), for illustrating our intention, we have extracted the following textual content:

... Per recobrar maniar Ad home cant a perdut lo maniar prin de l erba blanca ...

This same text has been segmented and marked up in three ways: its physical structure S1 (the manuscript is organized into pages, on two columns composed of lines), its syntactic structure S2 (the manuscript is composed of sentences and words) and its semantic structure S3 (the manuscript describes medical prescriptions which have signs, ingredients, plants and effects). The result is shown in Figure 2 where three XML documents hierarchically organize the same text according to these three different points of views.

Each structure marks up the text differently from another (see Figure 3). For example the segment of text ‘Ad home cant a perdut lo maniar’ is tagged by Sign in S3, while it is tagged differently in S1: ‘Ad home cant a perdut’ is marked by Line whereas ‘lo maniar’ begins another textual segment marked also by Line. Moreover notice that a Sign can overlap two lines. Therefore, these documents are independent with potential overlapping.

Structures (but not the relations between them) could be defined by means of a grammar like a DTD, an XML schema [6] or a RelaxNG schema [17].

These XML documents are currently considered separately even if researchers identify correlations between them. In particular, they would like to be able to express the following queries: What are the sentences that follow the signs? (this query combines syntactic and semantic structures); What is the prescription that contains the larger number of lines? (this query combines physical and semantic structures); What are the words cut by an end of line? (this query combines physical and syntactic structures).

Usual XML manipulation languages do not support concurrent structures, thus our issue is (1) to model multiple hierarchical structures in order to query them in a concurrent way and (2) to provide an XML environment to support multistructured documents.

Finally, notice that the relationship (eventually weak) between parts of these three structures are not defined: for example, one can remark that a page (in the physical structure) always begins with a prescription (in the semantic structure). It could be useful to use this kind of constraint during querying and to check consistency between structures. We tackle this issue by defining a schema to describe the relationships between parts of the concurrent structures.

1.2 Objectives

Our objectives are the following:

A suitable – XML compatible – data model. This model dedicated to multistructured textual documents is called MSXD. It enables (1) to consider several segmentations of the same text, and (2) to define a hierarchical structure for each of these segmentations. Notice that the structures could be weakly coupled and that there is no main structure. The set of segmentations and the set of hierarchical structures are deduced automatically from the given XML structured documents (for example produced by philologists). That is why they could be developed in a distributed way. The replication of the common text (in each XML document) can be seen as a drawback but (i) the data storage according to the volume is not really a problem, and (ii) synchronisation is not an issue because the common text is never updated: the only changing part (the structure) is not replicated. Moreover, each user can then edit its own copy offline. This data model is based on the use of hedges [28], the foundation of grammar language RelaxNG [17] (REgular LAnguage description for XML New Generation);

An extension of XQuery [7]. Our objective is to query the structures and the textual content in a concurrent way and to use an as much as possible unchanged XQuery. One of the original contribution of our proposal is that the MultiStructured document is never instantiated. As we said before, the set of structures can be syntactically described as a distributed set of XML documents: we want to keep each structure safe to use available XML tools and languages for its construction and its manipulation. Given a set of XML documents over the same textual value, a MSXD instance can be seen as an index over the textual value and between all the structures. This index is used to evaluate queries using concurrent structures;

The description of relationships between structures. We propose a solution to define a multistructured schema in order to describe the relationships (as weak constraints) between parts of the concurrent structures. This is done as a set of rules by means of Allen’s relations to constrain the relative position of fragments in the structures. This work is a first step towards validation across multiple hierarchies [20].

This paper is organized as follows: Section 2 defines the MSXD data model and shows the XML syntax associated to an MSXD instance. Section 3 presents an extension of XQuery to multistucture. Section 4 proposes a schema enabling to express constraints between concurrent structures. Section 5 specifies the MSXD architecture. Section 6 is dedicated to the related works and section 7 concludes. Currently, we only consider the definition of multiple structures in the single textual modality.
2 The MSXD model

We first choose to define the notion of a multistructured document and then to introduce all the concepts used in this definition. We illustrate the MSXD model with our running example. Our data model is based on the use of hedges (the foundation of RelaxNG). Informally, a hedge is a sequence of trees. In the XML terminology, a hedge is a sequence of elements possibly intervened by character data; in particular, an XML document is a hedge [28].

2.1 Formal model definition

Definition 1. A Multistructured document is a triplet \( (V, I, S) \) where \( V \) is a textual value, \( I \) a set of segmentations of \( V \) and \( S \) a set of structures associated to segmentations from \( I \).

A multistructured document can be seen as a textual value augmented with different hierarchical structures defined over it. These structures share the same text but concern (in general) different strings extracted from that text.

Definition 2. A segmentation of the textual value \( V \) of length \( l \) is a list \( X_V \) of strings such that \( X_V = \{x_i| x_i = V[b_i..e_i]\text{ and } b_0 = 0 \text{ and } e_i \geq b_i \text{ and } b_i = e_{i-1} + 1 \text{ and } e_{X_V[-1]} = l - 1\} \). \( b_i \) and \( e_i \) are respectively the start and the end positions of the fragment in \( V \). We define two functions for each \( X_V[i] \), \( \text{start}(X_V[i]) = b_i \) and \( \text{end}(X_V[i]) = e_i \). The set \( X_V \) makes a total partition of the textual value \( V \).

For our example, \( V \) is the text extracted from the manuscript (“...Per recobrar maniar Ad home cant a perdut lo maniar prin de l erba blanca...”) and we consider three segmentations \( X^1_V, X^2_V \) and \( X^3_V \) (corresponding to the set of textual contents of the XML elements from each structure \( S_1, S_2 \) and \( S_3 \), see Figure 2):

- \( X^1_V = x^1_1 \ldots x^1_4 \), with in particular \( x^1_1 = "Per recobrar maniar" \) and \( x^1_4 = "blanca..." \),
- \( X^2_V = x^2_1 \ldots x^2_{15} \), with in particular \( x^2_1 = "Per" \) and \( x^2_{15} = "blanca" \),
- \( X^3_V = x^3_1 \ldots x^3_4 \), with in particular \( x^3_1 = "Per recobrar maniar" \) and \( x^3_4 = "erba blanca" \).
We use the concept of fragment to define structures: fragments mark up a segmentation. Fragment positions in the textual value are useful to compute their relative positions.

**Definition 3.** A fragment \( f \) is defined over a segmentation \( X_f \) of the textual value \( V \) and an alphabet \( \Sigma_f \), a set of labels:

1. \( f = \varepsilon \) (the empty fragment), 
   \[
   \text{start}(f) = \text{end}(f) = 0
   \]
2. \( f = v_i \) with \( v_i \in X_f \), 
   \[
   \text{start}(f) = \text{start}(X_f[i]),
   \text{end}(f) = \text{end}(X_f[i])
   \]
3. \( f = v \prec x \) with \( v \in \Sigma_f \) and \( x \) a fragment, 
   \[
   \text{start}(f) = \text{start}(x),
   \text{end}(f) = \text{end}(x) \quad (f \text{ is called a tree})
   \]
4. \( f = xy \) with \( x \) and \( y \) two fragments and 
   \[
   \text{start}(y) = \text{end}(y) + 1,
   \text{start}(f) = \text{start}(x), \text{end}(f) = \text{end}(y).
   \]

Notice some remarks about definition 3:

- Rule 1 defines the empty fragment. Recall that we want to represent several segmentations of the same text in order to define a hierarchical structure for each of these segmentations (producing several structured documents over the same text). Using empty fragments (as milestones) is not compatible with our model because a fragment has to be related to a segmentation. The only legal empty fragment is the one associated with the empty document.

- Rule 3 uses the alphabet \( \Sigma_f \), which is a set of labels for fragments having a tree-like structure corresponding to XML element names: for the physical analysis, 
  \[
  \Sigma_f^1 = \{ \text{Manuscript}, \text{Page}, \text{Column}, \text{Line} \},
  \]
  for the syntactic analysis, 
  \[
  \Sigma_f^2 = \{ \text{Manuscript}, \text{Syntax}, \text{Sentence}, W \},
  \]
  for the semantic analysis, 
  \[
  \Sigma_f^3 = \{ \text{Manuscript}, \text{Prescription}, \text{Ingredient}, \text{Sign}, \text{Plant} \}.
  \]

- Rule 4 produces sequences of fragments. We do not make a distinction between a fragment and a singleton sequence containing that fragment and we do not consider nested sequences.
We now give some examples of fragments respectively constructed over segmentations \(X^1\) and \(X^2\):

- Over \(X^1\) (two fragments):
  (1) \(f_1 = x^1\) with \(x^1 = \text{"Per recobrar maniar"},\) 
  \[\begin{align*}
  \text{start}(f_1) &= 1, \text{end}(f_1) = 19 \text{ and} \\
  \text{start}(f_2) &= \text{Line} < x^2_1 >, \text{start}(f_2) = \text{start}(f_1), \\
  \text{end}(f_2) &= \text{end}(f_1);
  \end{align*}\]

  The fragment \(f_1\) is represented in XML syntax by
  
  \[
  \text{<Line>Per recobrar maniar</Line>}
  \]

- Over \(X^2\) (three fragments):
  (1) \(f_3^3 = x_3\) with \(x_3 = \text{"erba blanca"},\) 
  \[\begin{align*}
  \text{start}(f_3^1) &= 60, \text{end}(f_3^1) = 70 \text{ and} \\
  \text{start}(f_3^2) &= \text{Plant} < f_3^2 >, \text{start}(f_3^2) = \text{start}(f_3^1), \\
  \text{end}(f_3^2) &= \text{end}(f_3^1) \text{ and} \\
  \text{start}(f_3^3) &= \text{Ingredient} < f_3^3 >, \text{start}(f_3^3) = \text{start}(f_3^1), \\
  \text{end}(f_3^3) &= 92\) (we supposed that 92 is the end position of the textual fragment marked by "..." in our example);

  The fragment \(f_3\) is represented in XML by
  
  \[
  \text{<Ingredient>}
  \text{plant>erba blanca</plant>...}
  \text{</Ingredient>}
  \]

**Definition 4.** A structure is a tree \(f\) (a labelled fragment) over a segmentation of the textual value \(V\), \(\text{end}(f) = |Xf| - 1\) and \(\text{start}(f) = 0\).

Figure 4 illustrates our model over the two \(X^1\) and \(X^2\) segmentations (we do not show the third segmentation to make the reading of the figure easier) and the two structures \(S1\) and \(S3\) defined over them. The figure is composed of two parts: the first indicates the segmentations (start and end positions are associated to each textual segment inside a segmentation; for convenience the numbering of start and end positions only takes into account the segments used in our example), and the second part shows the hierarchical organization of fragments into a structure (physical structure on the left, semantic structure on the right).

In summary, from the XML documents of Figure 2, we can extract the text \(V\), we can deduce three structures (\(S1, S2\) and \(S3\)) built on three segmentations (\(X^1, X^2\) and \(X^3\)). Fragments are constructed over segmentations.

### 2.2 Relative position of two fragments

Our model is designed so that Allen’s relations [2] can be used on fragments in order to calculate their relative position inside a segmentation or between two segmentations.

**Definition 5.** Predicates on two fragments \(f_1\) and \(f_2\) are defined over one or two segmentations on the same textual value:

- before \((f_1, f_2)\) \(\equiv\) finishes \((f_2, f_1)\) \(\equiv\) end \((f_1) <\) start \((f_2)\)
- before \((f_1, f_2, n)\) \(\equiv\) finishes \((f_2, f_1)\) \(\equiv\) start \((f_2) -\) end \((f_1) = n\)
- meets \((f_1, f_2)\) \(\equiv\) met-by \((f_2, f_1)\) \(\equiv\) start \((f_1) =\) end \((f_2)\)
- during \((f_1, f_2)\) \(\equiv\) contains \((f_2, f_1)\) \(\equiv\) start \((f_1) >\) start \((f_2)\)
- overlaps \((f_1, f_2)\) \(\equiv\) is-overlapped \((f_2, f_1)\) \(\equiv\) start \((f_1) <\) start \((f_2)\)
- start \((f_1)\) \(\equiv\) started-by \((f_2, f_1)\) \(\equiv\) start \((f_1) =\) start \((f_2)\)
- finishes \((f_1, f_2)\) \(\equiv\) finished-by \((f_2, f_1)\) \(\equiv\) end \((f_1) =\) end \((f_2)\)
- equals \((f_1, f_2)\) \(\equiv\) start \((f_1) =\) start \((f_2)\)

Notice that if \(f_1\) and \(f_2\) are defined on the same segmentation the predicates meets and overlaps are always false.

Finally, we need to compute the level of a fragment in a structure. This level captures the parent/child relationship between two fragments in a structure.

**Definition 6.** Let \(F(s)\) \((s\) is a structure\) be the set of fragments \(f\) such that \(f = s\) or \(\exists x \in F(s), \exists a \in \Sigma_X | x = a < f >\). The function level \((s, f)\) returns the level of the fragment \(f\) in the structure \(s\), it is calculated with the following algorithm:

\(- \text{level}(s, s) = 0\)
\(- \text{level}(s, y) = \text{level}(s, x) + 1\) with \(x = a < y >\) \((x \text{ and } y \in F(s))\).

Figure 4 also shows the relative position of two fragments in two segmentations. In particular, the two following Allen’s predicates are true (they indicate that a \(\text{Plant}\) overlaps two \(\text{Lines}\)):

\(\text{overlaps(Plant(s5, e5), Line(s6, e6)) = true}\)

Lastly, notice that in Figure 4, we have associated to each fragment its level in the hierarchical structure to which it belongs. For example, in \(S1\),

\(\text{level}(S1, \text{Manuscript} <>) = 0\)
\(\text{level}(S1, \text{Page} <>) = 1\)
\(\text{level}(S1, \text{Column} <>) = 2\)
\(\text{level}(S1, \text{Line} <>) = 3\).

### 2.3 MSXD XML syntax

We see that for a given multistructured document, each structure can be described using an XML syntax, thus its schema can be described using RelaxNG (see Figure 5). We choose RelaxNG because our model relies on hedges
to represent XML documents and the design of RelaxNG is based on this theory [28]. Another equivalent solution could have been to use XML Schemas.

We define an XML syntax for an MSXD instance (see Figure 6). In this figure, we refer to the textual value of the manuscript (identified by an uri), each structure (identified by an uri) and its associated schema. Notice that segmentations are implicit.

3 Querying MSXD instances

Our data model is close to XDM (XML Data Model) [23] used in XQuery and XPath. Recall that XDM defines unnested sequences of items (nodes or atomic values). For querying, we propose an extension of XQuery to deal with a multistructured document.

Indeed, we propose to define an XQuery item as an atomic value or a fragment (instead of a node) and then we still deal with sequences of items. Thus, the XQuery language can be adapted to query a multistructured document. A XQuery on an MSXD instance with a single structure is equivalent to the same XQuery on the XML document corresponding to this structure. In the case of multiple structures, we extend the semantics of the filters of XQuery. For that, we successively study and extend:

- The accessors defined in the XQuery Data Model (XDM) [23]. Accessors can be seen as a set of primitives to access an instance of the XML data model (parent, child, ...).

- The axis defined in XQuery [7]. Axis are an higher level access mode to instances of the XML data model. An axis can be an accessor (for instance parent and child) or can be based on accessors (for instance the axis ancestor is defined by the recursive application of the accessor parent).

- The normalization of an extended XQuery into a core XQuery (as defined in the XQuery Formal semantics [18]). To formally define the semantics of XQuery, a subset of XQuery named XQuery core has been defined. Every XQuery can be expressed in the core. For instance, an XPath step is translated in a For Let Where Return (FLWR) expression. The dynamic environment existing during the evaluation of an XQuery is explicitly defined by binding a set of standard variables in the XQuery core.

Our objective is to use an as much as possible unchanged XQuery, that is why we choose to rely on the XQuery normative documents.

3.1 Extending XDM accessors and XQuery axis

To navigate in every structure and to adapt Allen’s relations, we choose to slightly modify the semantics of some accessors or operators (close to Allen’s relations). In other cases, we define new functions.

Accessors. First, to express the containment we adapt the parent/child relationship: The dm:children and dm:parent accessors have been extended to return every parent and every child of a fragment in every structure it belongs to.

Axis. As a fragment does not belong to every structure and as we want them to be as general as possible, the axis ancestor and descendant cannot be defined by applying recursively the accessors dm:parent and dm:children as it is done in XQuery2. Thus, we define

\[ \text{http://www.w3.org/TR/xquery/id-full-axis-feature} \]
<!-- Physical Structure -->
start =
    element Manuscript {
        element Page {
            element Column {
                element line {
                    text+
                }+
            }+
        }+
    }

<!-- Syntactic Structure -->
start =
    element Manuscript {
        element Page {
            element Column {
                element line {
                    text+
                }+
            }+
        }+
    }

<!-- Semantic structure -->
start =
    element Manuscript {
        element Prescriptions {
            element Prescription {
                | text | Dosage | Effect | Ingredient
                | AdministrationMode
                | element Sign { text }
                | element Concoction {
                    | text | Effect
                    | AdministrationMode
                    | element Action { text | Ingredient }+ 
                }+
            }+
        }+
    }

Effect = element Effect { text }
Sign =
    element Ingredient {
        | text | Dosage
        | element Mineral { text }
        | element Plant { text }+
    }
AdministrationMode = element AdministrationMode { text }
Dosage = element Dosage { text }

Figure 5: Schemas for the Physical (S1), Syntactic (S2) and Semantic (S3) structures

<MsXmlDoc xmlns="http://lsis.univ-tln.fr/msxd/doc/v1/">
<TextalValue uri="http://lsis.univ-tln.fr/msxd/value/manuscript"/>
<Structure type="http://lsis.univ-tln.fr/msxd/structure/manuscript/physical" uri="http://lsis.univ-tln.fr/msxd/instance/S1.xml"/>
</MsXmlDoc>

Figure 6: XML syntax for the multistructured manuscript
two new accessors: `dmmsxd:ancestor` and `dmmsxd:descendant` which respectively return the sequence of fragments containing a fragment or contained in a fragment (according to start and end positions).

**Order.** We need to consider Allen’s relations used to compute relative position between two fragments inside a segmentation or between two segmentations. A partial order on fragments in a given multistructured document can be defined on start and end positions. As in XQuery, the boolean operators \( n_1 << n_2, n_1 >> n_2 \) and \( n_1 = n_2 \) can be defined. They are respectively true for the fragments \( n_1 \) and \( n_2 \), if \( n_1 \) is before \( n_2 \), \( n_1 \) is after \( n_2 \) and if \( n_1 \) and \( n_2 \) are the same fragment. We use the two first boolean operators \( (<<, >>) \) to express the following (after) and preceding (before) Allen’s relations.

**New functions.** We introduce each of the remaining relations as functions (evaluated according to the relative positions of two fragments). We only give here the definition of two Allen’s relations \( (\text{equals} \text{ and} \text{overlaps}) \) and the corresponding operators because they are used in the query examples given below. The remaining relations can be defined in the same way:

- the function `msxd:is-equal(n_1,n_2)` and the operator `is-equal`, are true if \( n_1 \) and \( n_2 \) have the same start and end positions in the same multistructured document,

- the function `msxd:is-overlapping(n_1,n_2)` and the operator `is-overlapping`, are true if \( \text{start}(n_1) < \text{start}(n_2) \text{ and } \text{end}(n_1) > \text{start}(n_2) \text{ and } \text{end}(n_1) < \text{end}(n_2) \).

### 3.2 Extending the dynamic evaluation

In the formal semantics of XPath and XQuery, the dynamic context is explicitly defined by binding variables. In particular, \$fs:sequence, \$fs:dot, \$fs:position and \$fs:last variables respectively represent the sequence of context items, the context item, the context position, and the context size. The side effect of each operator is also explicit in the core XQuery.

It is necessary to extend the dynamic context to carry information about every structure associated with an MSXD instance. We extend it by binding a new variable `\$msxd:selected_structures` to a sequence of strings that represents the set of ids of the structures to be taken into account during the evaluation of the query.

In our example, when the document is loaded by default:

\[
\text{$msdoc := msxd:doc("manuscript.msxd")}$
\]

We need to define two basic functions to return existing structures in a document and to create the instance of a document:

- `msxd:structures($arg as fragment()* as xs:string*)` returns a sequence of strings which represents the ids of structures to which every fragment of the sequence \$arg belongs.

- `msxd:doc($uri as xs:string?) as document-node()` retrieves the XML description of an MSXD instance using \$uri and returns its document fragment \$root (equivalent to fn:doc() in http://www.w3.org/TR/xpath-functions/).

This changes the dynamic context by binding `\$msxd:selected_structures` to `msxd:structures($root)`, ie by default every structure of a document are consired during a query.

Finally, we slightly modify the normalization of a path expression in XQuery (a step followed by a relative path expression) to return only fragments of the selected structures. The new rule is given in Figure 7.

The standard normalization transforms an XPath step into a FLWR expression and it sets the dynamic environment after the evaluation of the first step, and for each item of the result \( (\text{$fs:sequence$}) \) the relative path is evaluated (Lines 3,4 and 5). In our extension, we restrict the results to items from the selected structures (Lines 6, 7 and 8). Lines 1 and 2 ensure that each fragment is unique and that every fragment is sorted according to the document global order.

### 3.3 Query examples

We propose some queries for our running example:

**Prolog - Binding the multistructured document to a global variable**

```
declare variable
    $msdoc := msxd:doc("manuscript.msxd")
```

**Q1 – Children of Manuscript**

```
$msdoc//{{/Manuscript/*
```

returns every child of the fragment Manuscript which is shared by every structure (the results is the sequence Page from S1, Syntax from S2 and Prescriptions from S3, see Figure 2).

**Q2 – First Sentence of Prescriptions described on one Column**

```
$msdoc//{{/Column//Prescription//Sentence[1]
```

**Q3 – Words cut by an end of Line**

for \$v in $msdoc//Line

return $msdoc//{{[ . is-overlapping $v]}

This variable can be used to restrict the set of structures. If it is set to a subset of the structures, only this subset is taken into account by the XQueries.
A DATA MODEL AND AN XQUERY EXTENSION FOR... Informatica 35 (2011) 141–156

Figure 7: Extended normalization of a Path expression

Q4 – Columns which are Sentences
for $v$ in $msdoc//Sentence
return $msdoc//Column[. is-equal $v]

Q5 – Sentences containing Plant
$msdoc//Sentence[descendant::Plant]

Q6 – First Sentence after a Sign
$msdoc//Sign/following::Sentence[1]

Q7 – First Sentence after a Sign
for $h$ in $msdoc//Sign
return $msdoc//Sentence[. >>> $h][1]

is the same as Q6 but using the order operator instead of the following axis.

Q8 – Children of Manuscript in S1
let $msxd:selected_structures :=
"http://lsis.univ-tln.fr/msxd/instance/S1.xml"
return $msdoc/Manuscript/*

is the same as Q1 but returns only children from S1 because the variable $msxd:selected_structures is explicitly set to the identifier of S1.

4 A Schema for multistructured documents

We define a schema for multistructured documents as a set of rules (vs a content model definition) because our structures are weakly coupled and the multistructured document is not hierarchical. Allen’s relations (starts, overlaps, equals, ...) enable to constrain the relative position of fragments belonging to different structures. The constrains are expressed using XPath based predicates, we suppose that an XPath expression applied to a structure returns a sequence of fragments.

Definition 7. A Multistructured document schema is a pair 
$(G_3, C)$ where $G_3$ is a set of grammars defining valid structures and $C = \{c | c = c[p_1 \text{ in } s_1, p_2 \text{ in } s_2] \}$ is a set of constrains, where $c$ is the name of an Allen’s predicate and $p_1, p_2$ are XPath expressions applied to the structures $s_1$ and $s_2$. The constrain is true if for each fragment $f_1$ in $\operatorname{val}(p_1)$, it exists a fragment $f_2$ in $\operatorname{val}(p_2)$ such that $c(f_1, f_2)$ is true. A document is valid according to the schema if and only if every constrains in $C$ are true.

Figure 8 (see comments in the figure) shows an XML syntax for multistructured documents schemas and illustrates some constrains between fragments of the three structures of our running example (notice that each constrain is applied to two structures). Every constrain could be read in the same way, for example

– Rule 1: Root fragments of physical and syntactic structures are equal. Each fragment matching /Manuscript in every document valid according to the structure (whose alias is) manuscript_physical must be equal to at least one fragment matching /Manuscript in every document valid according to the structure (whose alias is) manuscript_semantic;

– Rule 3: A page starts by a prescription. Each fragment matching Page in every document valid according to the structure (whose alias is) manuscript_physical starts by at least one fragment matching Prescription in every document valid according to the structure (whose alias is) manuscript_semantic;

– Rule 4: A prescription contains sentences. Each fragment matching /Prescription in every document valid according to the structure (whose alias is) manuscript_semantic contains at least a fragment matching sentence in every document valid according to the structure (whose alias is) manuscript_syntactic.

This work is a first step towards validation across multiple hierarchies [20]. It enables to check the conformance of concurrent annotations attached to the same textual document related to predefined weak relationships between parts of different structures. Even if the validation is optional, it is useful to use this kind of constraints in case of distributed annotation to check the consistency of structures before querying.
Figure 8: A grammar for our multistructured document
5 Implementation of the MXSD data model and query language

In this section, we describe the architecture of an implementation of the MSXD data model and of an XQuery engine extended to support multistructured documents. Figure 9 presents the functional architecture of our prototype and a typical use case in four steps. The figure is organized in two lines and two columns. The first line presents the user level and the second the MSXD engine level. The first column describes the design and the indexing an MSXD instance from a set of XML documents, the second column illustrates the querying.

5.1 User space: An almost usual XML environment

The user space (parts 1 and 3 in Figure 9) is close to an usual XML environment. First, each user can describe its structures in XML documents which mark up the same text (see Figure 2). Notice that the structures could be distributed over a network. Then the user defines a (virtual) MXSD instance with respect to MSXD XML syntax (see Figure 6). To do this the user only needs an XML editor, he does not have to change its habits.

To query an MXSD instance, a user can express it using the language defined in Section 3. The indexing is automatically done if needed (see next section). Recall that the language extends XQuery: if the user queries a single structure, it is the standard XQuery. Query result is a sequence of fragments represented in XML.

5.2 MSXD Engine: indexing and querying structures and content

As we have shown in Section 2, an MSXD document is a XML meta-document which refers to one XML document for each structure (both of them sharing the same textual value); our implementation of the data model can be seen as a dynamically built index between them (part 2 in Figure 9). We choose to separate the indexing of the structure and the indexing of the content.

The analysis of the first structure enables us to deduce the textual value. The textual value is indexed only once in a specific component which is in charge of the textual indexing (to answer full text queries) and in charge of the access to the textual values of fragments (to build XML answers to the queries). When the other structures are analyzed, the consistency of the text is checked, and an alignment by means of spaces, tabulations or end of line is automatically processed if necessary. For the storage of the textual value and its indexing, we use Apache Lucene.

The analysis of each structure enables the creation of a relational representation of the structures. We define three main relations for a given document to store: (1) the fragments with their node type (element, attribute, text, ...), their start and end positions in the normalized textual
value, their label, (2) the structures and (3) the structure contents (id of the structure, id of the fragment, level of the fragment in the structure). Notice that this representation is independent from the textual value, it uses start and end positions (the textual value manager stores real values) and it computes the level of fragments into each structure (a fragment can then be shared between structures). In our prototype, we choose to embed a java RDBMS\(^4\) but an external one can also be used.

We define an API consisting in usual DOM [1] API accessor\s extended with operators based on Allen’s relations (as defined in definition 5). We store segmentations\s as tuples in a RDBMS, so we implemented it in SQL. This API provides a high level access to multistructured documents. MSXD instances conform to the DOM API and provide new methods such as the access to overlapping fragments. The query language prototype relies on this API.

To implement the query engine (part 4 in Figure 9) for this academic prototype, we choose to work step by step and to use standards. First, we translate the user query in the XQuery Core language\(^5\). Even if it is not designed to be the foundation of prototypes, we choose it because we need a clean “simple” language with a well defined semantics. Then, the XQuery Core query is used to build a query tree, which is optimized before its evaluation. Most of our operators are implemented to work in pipe line, the XQuery filters operators (operators which give access to children, ancestors, ... of a given fragment) use the extended DOM API. Notice that, the SQL translation remains visible at query time for future optimization (for example by grouping several SQL queries nested with FLWR operators into a single SQL query).

Figures 10 and 11 show two screenshots of the main window of the application and of the query tree display window. Figure 10 shows a capture from our prototype during the evaluation of query Q2. A user can select (from a set of test cases) or edit an extended XQuery (top/right). The automatic translation in XQuery Core is shown bottom/right and the result (either in XML or in internal format by means of start and end positions) is displayed at bottom/left.

The second figure displays the query tree corresponding to the core query and displays dynamic information during the execution (number of items created or filtered by operators, ...). Finally, we developed an implementation of the multistructured schema validation where constraints are checked sequentially. In order to obtain a more efficient validation and to provide a more intuitive way to express constraints, we are investigating the use of ontologies. We currently tackle this problem with a linguistic application of multilevel analysis of multimodal data (OTIM—http://lpl-aix.fr/~otim).

### 6 Related works

If we look at XML standards, it seems clear that the standard tree-like model\(^2\) and namespaces\(^9\) could be used to represent multistructured documents if each structure is hierarchical and can be merged with others. But, this is not the case in general. In our example, some elements from the physical and syntactic structures can overlap (Line and Sentence). The problem of overlapping is not recent see [20] for a review. Several works have studied multistructured documents in the context of XML for document-centric encoding. We classify the main proposals into three categories.

The first one concerns the very first works about the representation of several hierarchical structures inside the same text (the CONCUR feature [24] of SGML, TEI\(^6\)); these solutions are often syntactic. TEI’s solutions need to choose either a flat representation of the multistructured document or a main (hierarchical) structure and to use references (ID/IDREF) for the description of the other structures.

The second category is based on proprietary graph models. LMNL\(^7\) proposes a new markup language and model such as to overcome the limitations of hierarchical markup in XML and to get an instance of a multistructured document. LMNL graph-based model is not XML compatible even if it is able to import/export. Notice that LMNL considers user annotations but no solution for querying. MultiX [14] is a proprietary graph-based model. It is possible to serialize an instance of it in an unique XML document. The multistructured querying is achieved by means of a set of XQuery [7] functions which in particular, deals with overlapping. Based on [14], [29] proposed a methodology for the construction of multistructured documents. This high level approach aims at defining structures during the construction process. To our knowledge, none other contribution considers a priori the problem of that construction which leads to restructuring and automatic differentiation of structures. We did not yet consider the problem of multistructured document edition.

MVDM (Multiview Document Model) [21] is a proprietary graph-based model. The model aims at considering multimedia documents and therefore at representing different kind of relationship between two document entities (and not only hierarchical relation). MVDM focuses on the notion of view which corresponds to a particular organization of a document. Stored in a document repository, multistructured documents can be queried according to criteria linked to one or several views of that document (automatic generation of SQL queries taking into account overlapping); another solution to navigate in the repository is proposed with a multidimensional analysis (OLAP).

At last, Annotation graphs (AG) [5] are coming from the linguistic domain. Annotation graphs propose a proprietary formal model for the representation over the same flow (au-

---

\(4\)http://www.postgresql.org/

\(5\)http://www.w3.org/TR/xquery-semantics/

\(6\)http://www.tei-c.org/P4X/NH.html

\(7\)http://www.lmnl.net/
Figure 10: Evaluation of XQuery Q2 in our prototype

Figure 11: Tree of the XQuery Core for Q2 in our prototype
dio, text, …) of several structures (may not be hierarchical). An XML “flat” representation of an AG is proposed but no solution for querying.

The third category presents XML compatible contributions. A very interesting framework is proposed in [19]. It is a new model based on the Goddag’s data structure [31] which can be seen as a generalization of DOM trees for the representation of multi hierarchical XML documents. This proposal defines also an extension of XPath [26] to navigate between different structures sharing the same textual data (with a specific axis for concurrent querying such as over ‘ancestor’ or ‘descendant’). Another proposal, the colored trees [27], deals with multiple hierarchies in a data-centric context. It aims at sharing atomic data and it does not consider overlapping thus it is out of our scope. The idea is to build several hierarchies (called colors) over the same set of values (text nodes). Thus, nodes are multicolored. In order to navigate between colors/hierarchies, the authors extend the notion of step in XPath. A step begins by a choice of a color (and thus of a structure) before the usual selection of an axis, a test node and some predicates.

Figure 12 summarizes the main features of each proposal related to multistructured documents according to some criteria: Data model, XML compatibility, Existence of a main structure (the user at the logical level or the system at the physical level chooses a main structure so others structures have to be set with regard to this main structure), Validation of a multistructured document (is it possible to define a schema for multistructured documents validation across multiple hierarchies [20]) Operators (for querying several structures and content in a concurrent way). No query language has been defined for querying annotation graphs, but it exists operators that complete the XML propositions (XPath in particular [3, 4]), they are related to the linguistic context.

Our proposal belongs to the third category, our objective is to remain close to XML standard. Our model is close to Goddag’s. Indeed, we want to keep the hierarchical aspect of each structure, so that classical XML tools remain available. But, Goddag does not provide mechanisms to add annotations (as LMNL does) and it does not describe relationships between structures for enabling validation across multiple hierarchies [20] (none of these proposals, whatever the approach is, proposes it). We do not detail user annotation but our proposal considers it (see [11] and [12]). Annotations represent textual data added by a user to the text in one structure and so missing in the other structures. It represents specific information that the user needed to integrate to its analysis.

For querying several hierarchies over the same textual content in a concurrent way, we chose to extend the semantics of the filter of XQuery. Our objective is not to add new axis (like Goddag) or to extend XPath step with colors (as with colored trees). Moreover, even if we propose every Allen’s relations, we choose to stay simple and to use an as much as possible unchanged XQuery by only adding the necessary function (Unlike MultiX).

7 Conclusion

In this paper, our intention was twofold. First, we defined a XML compatible model for multistructured textual documents which is based on the use of hedges (the foundation of RelaxNG). A multistructured textual document is a text which has several concurrent hierarchical structures defined over it. Each structure corresponds to an analysis of the text according a specific use. Secondly, we proposed an extension of XQuery in order to query structures and content in a concurrent way. We applied our proposals using a medieval manuscript text written in occitan. Finally, we describe the architecture of an implementation of the MSXD data model and of a XQuery engine extended to support
multistructured documents. Our solution is entirely XML compatible and conforms to standards.

A multistructured textual document is defined as a set of fragments defined on the same textual value and grouped in concurrent hierarchical structures. The key idea is to propose a method for a compact description of multiple trees over a single text based on segmentation. Segmentation encoding allows querying overlap/containment relations of markups belonging to different structures. We showed how each structure can be described in an XML document. The multistructured textual document is never instantiated.

To query a multistructured textual document, we chose to extend the semantics of the filter of XQuery. We show how to take into account equality, overlapping and other Allen’s relations. For that we added functions and operators to XQuery. We are trying to avoid changing the structure of XQuery (as colored trees did without considering overlapping). Moreover, we did not simply add a new axis (as goddag did by adding $\text{xancestor \ xdescendant}$), but one can notice that it makes the query easier to read. However, normalization of $\text{xancestor \ xdescendant}$ or even a new axis associated to Allen’s relations can be rewritten into our proposal.

Moreover, we defined a multistructured schema in order to express weak constraints between structures; it is defined as a set of rules, Allen’s relations are used to constrain the relative position of fragments in the structures. An alternative solution could rely on the use of ontologies. It could offer more flexibility. We currently tackle this proposal with a linguistic application of multilevel analysis of multimodal data (Project OTIM 8).

Our main perspective is the definition of multiple structures in other modalities than the single textual one. For example, it could be useful to define one or several structures associated to the image of a manuscript as it is done for its textual transcription. Then, the objective would be to manipulate the set of all these structures in a concurrent way. Secondly, we plan to extend our query engine to distribute parts of queries in a P2P network and to enable data sharing.

References


8http://lpl-aix.fr/~otim


