MODELING, ENCODING AND QUERYING MULTI-STRUCTURED DOCUMENTS

Pierre-Édouard Portier, Noureddine Chatti, Sylvie Calabretto,
Elöd Egyed-Zsigmond and Jean-Marie Pinon
Université de Lyon
LIRIS UMR 5205 – INSA LYON
7, avenue Jean Capelle
69621 Villeurbanne Cedex, France

ABSTRACT

THE ISSUE OF MULTI-STRUCTURED DOCUMENTS BECAME PROMINENT WITH THE EMERGENCE OF THE DIGITAL HUMANITIES FIELD OF PRACTICES. MANY DISTINCT STRUCTURES MAY BE DEFINED SIMULTANEOUSLY ON THE SAME ORIGINAL CONTENT FOR MATCHING DIFFERENT DOCUMENTARY TASKS. FOR EXAMPLE, A DOCUMENT MAY HAVE BOTH A STRUCTURE FOR THE LOGICAL ORGANIZATION OF CONTENT (LOGICAL STRUCTURE), AND A STRUCTURE EXPRESSING A SET OF CONTENT FORMATTING RULES (PHYSICAL STRUCTURE). IN THIS PAPER, WE PRESENT MSDM, A GENERIC MODEL FOR MULTI-STRUCTURED DOCUMENTS, IN WHICH SEVERAL IMPORTANT FEATURES ARE ESTABLISHED. WE ALSO ADDRESS THE PROBLEM OF EFFICIENTLY ENCODING MULTI-STRUCTURED DOCUMENTS BY INTRODUCING MULTIX, A NEW XML FORMALISM BASED ON THE MSDM MODEL. FINALLY, WE PROPOSE A LIBRARY OF XQUERY FUNCTIONS FOR QUERYING MULTIX DOCUMENTS. WE WILL ILLUSTRATE ALL THE CONTRIBUTIONS WITH A USE CASE BASED ON A FRAGMENT OF AN OLD MANUSCRIPT.

Keywords

Multi-structured document; XML; MultiX; Multi-structured Document Querying; XQuery

1 INTRODUCTION

1.1 DOCUMENT STRUCTURING

Document structuring is used in many applications such as document exchange, integration and information retrieval. Several types of structures (physical, logical, semantic …) (Nanard & Nanard, 1995) (Poullet, Pinon, & Calabretto, 1997) have been defined for several specific uses. Moreover, a document can actually be a vehicle for various media types that can themselves introduce other structural layers (such as the temporal dimension of an audio track).

A single document can be used in many contexts. Thus, its content might be presented through many structures. In this case, the structures are said concurrent or parallel, since they share the same content. Humanities provide numerous instances of such structures. For example, the study of medieval manuscripts often implies the creation of concurrent hierarchical structures. First, we can consider a ubiquitous and trivial case of overlapping: the physical book-structure of a manuscript (a sequence of pages, columns, lines, etc.) and its syntactical structure (a sequence of sentences, words, etc.). Less trivial would be a structure of the sequences of damaged characters. Figure 1 is an extract of such a medieval manuscript fragment with its transcription. It should be noted that damaged characters are overlapping with words and words are overlapping with lines. The emphasis on multi-structured documents comes with the possibility of formally encoding documentary structures with digital representations. The fact that we find numerous examples of multi-structured documents in the TEI (Text Encoding Initiative) guidelines (TEI Consortium, 2011) should prove it. Among those examples, we can mention in verse drama, the structure of acts, scenes and speeches that often conflicts with the metrical

structure. Indeed, poems often provide multiple concurrent structures. Poem 1 is an example of two stanzas from a poem by Lewis Carroll[1] with verses, speeches, and syntactic elements producing overlapping concurrent structures.

"What's the good of Mercator's North Poles and Equators,
Tropics, Zones, and Meridian Lines?"
So the Bellman would cry: and the crew would reply
"They are merely conventional signs!

"Other maps are such shapes, with their islands and capes!
But we've got our brave Captain to thank:"
(So the crew would protest) "that he's bought us the best—
A perfect and absolute blank!"

POEM 1 TWO STANZAS FROM THE HUNTING OF THE SNARK BY L. CARROLL

FIGURE 1 A FRAGMENT OF AN OLD MANUSCRIPT AND ITS TRANSCRIPTION

1.2 CONSTRUCTION OF MULTI-STRUCTURED DOCUMENTS

1.2.1 INTRODUCTION

The work presented here will deal with the central issue of encoding multi-structured documents. However, in order to make clear the advantages of our model over existing solutions, we now introduce the important but little-studied problem of the construction of multi-structured documents. Indeed, in a large majority of real life situations, documents are not *a priori* given but have to be constructed. Moreover this construction is in most cases the work of a team. Thus, it is to be expected that the partition of the annotations in a number of different structures comes from this collaborative work. In other words, the construction of multi-structured documents is a dynamic process. How do structures emerge? How to check on their coherence? … In order to tackle these important issues, we need an adequate encoding for multi-structured documents.

In a previous work (Portier & Calabretto, Creation and maintenance of multi-structured documents, 2009) (Portier & Calabretto, DINAH, a philological platform for the construction of multi-structured documents, 2010) we explicitly studied the construction of multi-structured documents. Although the formal representation we then used was based on RDF rather than XML, this previous work offers a well-adapted and non-trivial applicative context for the XML based model we now propose. Moreover it gives us the opportunity to motivate the key choices behind our proposition.

1.2.2 CONTEXT

We studied how multi-structured documents are constructed in a multi-users context composed of philologists. Our work is based on experience gained working with Humanities researchers building digital editions of large archives of (mainly handwritten) manuscripts from various epochs. Digital editing covers the whole editorial, scientific and critical process that

[1] http://www.gutenberg.org/files/13/13-h/13-h.htm

leads to the publication of an electronic resource. In the case of manuscripts, editing mainly consists in the transcription and critical analysis of digital facsimiles, that is to say, the creation of a textual document associated with the images of a handwritten manuscript. We discovered that multi-structured documents construction was at the heart of their work. Indeed, they need to let coexist a multiplicity of structures in order to access a document according to many interpretations. Thus, we proposed a methodology promoting the emergence of multiple structures in a multi-users context.

1.2.3 SCENARIO

This study gave us a lot of scenarios similar to the following: a philologist finds a consistent subset about medicinal plants in a stack of pages of consequent size. He creates a new collection from this subset. He creates an association named "mainSubject" between this collection and the topic "Medicine". He starts to transcribe the collection and annotates some intervals of the text with terms such as "quotation", "prescription", etc. Later, he may discover that this collection is in fact a preparatory work for another piece of the archive. He then creates an association named "preparationFor" between the two collections. Etc.

How is it that, for example, a user chooses to place the term "citationTitle" within a structure named "Citations" while he affects the term "line" to the "Physical" structure? This kind of question brought us to define a methodology for the construction of multi-structured documents.

1.2.4 METHODOLOGY

First of all, we only consider content addressable by concrete intervals: characters intervals in a text, time intervals in an audio or video document. In order to offer some unity to the various structures emerging from the work of a group of users, we introduce an *a priori* rule: a structure must form a hierarchy. In other words, the annotated intervals of a structure should never overlap. By dynamically checking the validity of this rule we managed to ease the collaborative construction of multi-structured documents. We should now briefly illustrate this idea with the previous example of an old manuscript (see Figure 1).

We assume that the researchers made use of annotation terms such as: "line", "w" (for word) and "dmg" (for damaged characters). The transcription process continues until a word is overlapping with two lines (see strong & dashed lines of Figure 2).

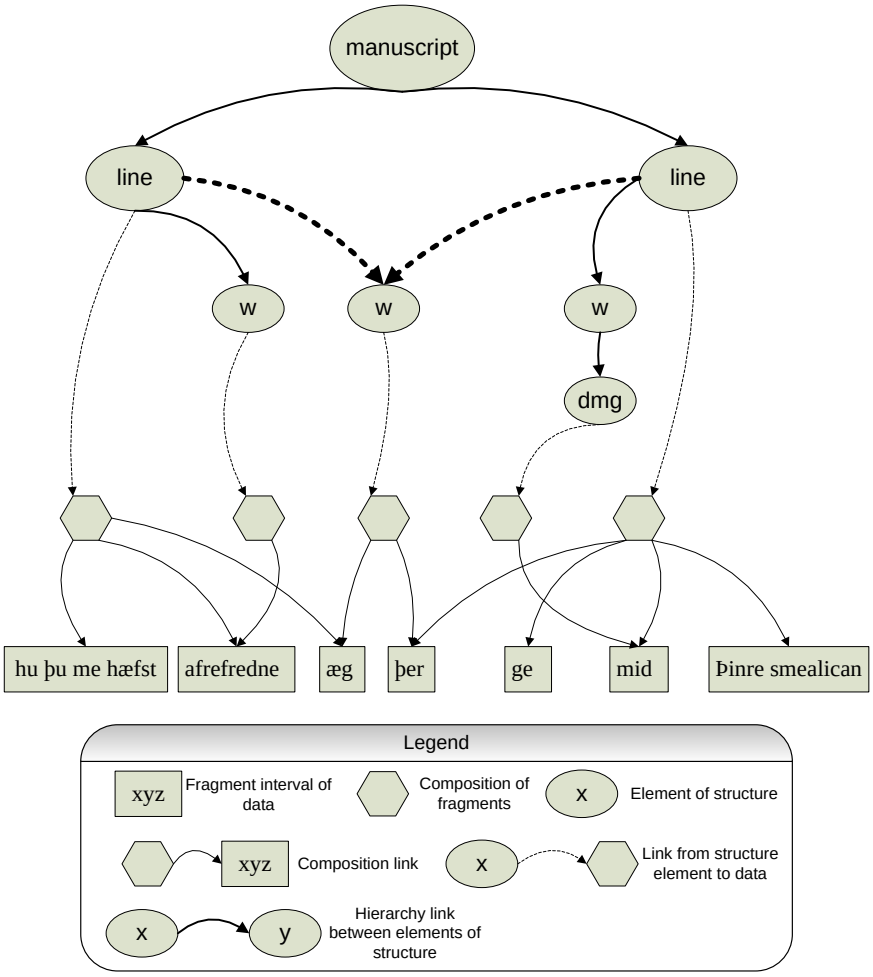

FIGURE 2 METHODOLOGY FOR THE CONSTRUCTION OF MULTI-STRUCTURED DOCUMENTS: RESTRUCTURING IS NECESSARY SINCE A WORD IS OVERLAPPING WITH TWO LINES

The user will then be alerted about an incompatibility between the terms “line” and “w”. Therefore the user is advised to re-organize the structures. The result of this operation might me similar to the one illustrated by Figure 3.

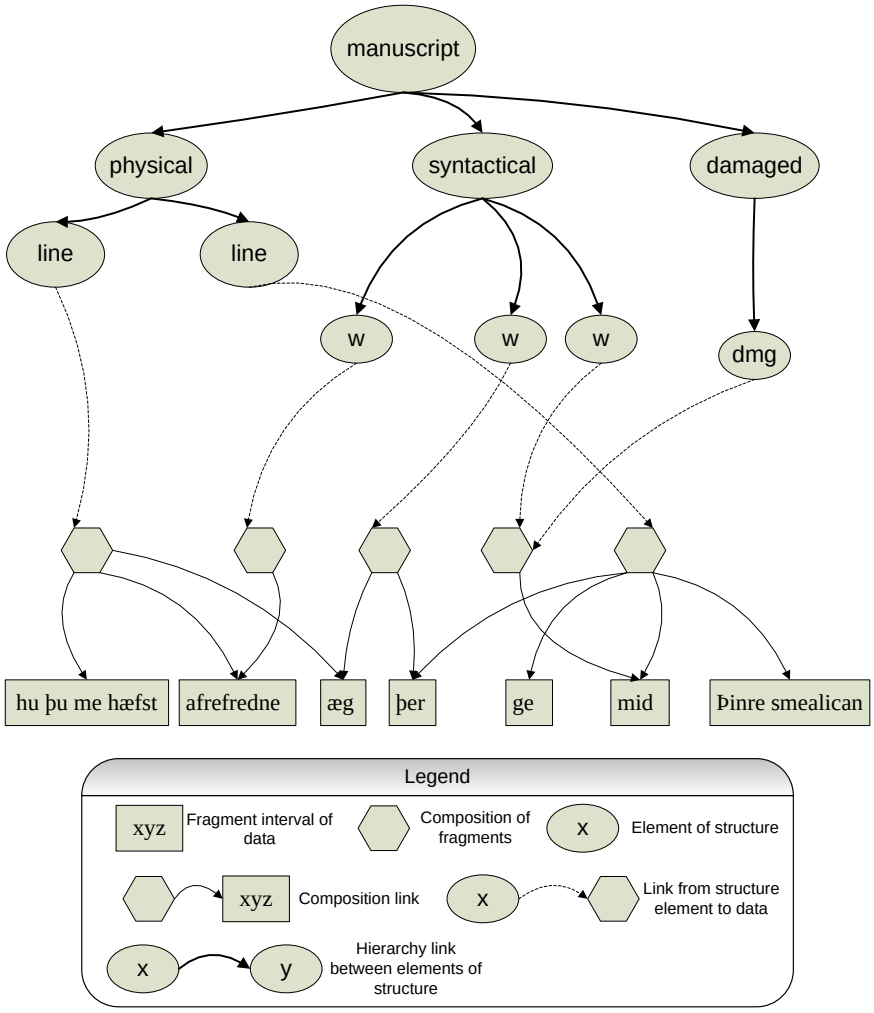

FIGURE 3 METHODOLOGY FOR THE CONSTRUCTION OF MULTI-STRUCTURED DOCUMENTS: A USER HAS DEFINED THREE NEW STRUCTURES (SYNTACTICAL, PHYSICAL AND DAMAGED)

1.2.5 CONCLUSION

Finally, this strategy for the management of multi-structured documents promotes the construction of a multiplicity of structures that should reflect the perspectives adopted by the users working on a documentary archive. Each user has the liberty to create new structures. Moreover, when overlapping structures are detected, users are encouraged to solve the problem by introducing a new structure.

From this specific applicative aspect of multi-structured documents construction, we can infer some requirements for the encoding of such documents. In particular, the fragmentation approach appears to be a key feature. First of all, as we can learn from the multi-structured documents examples given by the TEI, in a large majority of cases, structures are sharing fragments of the raw content. Also, and we are inclined to stress this point, some of the non-trivial applications of multi-structured documents seem to rely on a dynamic, and often collaborative, manipulation of documentary fragments. Indeed, multiple structures often appear when users are carrying out highly interpretative tasks that require a fine-grained analysis of the primary content. In these circumstances, the fragmentation is an all-pervasive operation that should be made as straightforward as possible: a structure should be able to combine any subset of fragments independently from other structures.

1.3 AGENDA

We have proposed (Chatti, Kaouk, Calabretto, & Pinon, 2007) a generic model called MSDM (Multi-Structured Document Model), in which we define the notion of a multi-structured document. In this paper we deal with the multi-structured documents encoding issue. Encoding (i.e. making explicit an interpretation of a text) is very often achieved through using markup languages (i.e. sets of markup encoding conventions). Therefore, XML as a metalanguage for the description of markup languages is the tool of choice for many document centric projects. Often, parallel structures are encoded separately in distributed XML files. Encoding these structures in separate files has many disadvantages, such as content redundancy and breaking interdependency. This kind of solution introduces significant document management problems.

To address these problems, it may be interesting to encode all structures in a single XML document (one file for all concurrent structures), by superimposing them to avoid content redundancy. As a matter of fact, it is very difficult to encode concurrent structures in a single XML file. Usually the overlapping problem makes it impossible to get a well formed XML document. XML is based on a tree model (and therefore elements cannot overlap). In this paper, we propose a new formalism called MultiX, which allows the efficient encoding of multi-structured documents. MultiX, which uses an XML syntax, is based on the MSDM model.

After a survey of related works (section 2), we present, in section 3, a formal description of the MSDM model and illustrate it with the previous example of a manuscript document. This same example will be used in section 4 to illustrate the MultiX formalism. Section 5 will be dedicated to the multi-structured document querying issue.

2 RELATED WORKS

2.1 INTRODUCTION

The problem of encoding concurrent structures has attracted much attention. Several approaches have been proposed to deal with this problem. We classify the main approaches into two categories. The first one contains the very first works on the representation of several hierarchical structures inside a same text; these solutions are often exclusively syntactical. The second category includes works grounded on well-defined formal models. In a previous survey, (Calabretto, Bruno, & Murisaco, 2007) was referring to three categories making a distinction between “XML-compatible” models and the others. We choose to consider XML-compatibility as one of the dimensions against which the models will be studied.

2.2 DIMENSIONS RETAINED FOR THE ANALYSIS

We analyze each solution against six dimensions:

- *Data model*: On what kind of model is the solution based? Is it a tree, a graph, something else?

  We should see that in most cases the solutions will be based on a tree, a set of trees or a generic graph. However, a few of them will use more exotic models. Moreover, there are the syntactical only solutions with no clearly defined data model. We consider the presence of a well-defined formal model as a most important feature. It is a quality criterion ensuring the continued evolution of the solution. Also, most of the time, this dimension will not be independent from others. Indeed, it can affect the query mechanisms, the possibility to update a multi-structured document, etc.

- *XML-compatibility*: Is the solution compatible with the XML formalism?

XML is nowadays the *de facto* standard for the representation of electronic documents. Among other things, the formalism is associated with numerous standard and efficient tools for processing documents: XSLT, XPath, XQuery, etc. Thus, XML-compatibility is essential. A solution doesn't achieve XML-compatibility by merely managing XML files but by allowing a straightforward use of the aforementioned tools. Regarding this last point, different approaches are encountered. Some of them modify the XML model itself; some modify the XPath or XQuery specification while others manage not to alter the XML core. We incline toward this last category since we consider solutions modifying the core of the XML model unsustainable compared to solutions taking advantage of the "galaxy" of XML tools with light, nonintrusive, approaches.

- *Master structure*: Does the solution rely on a master structure or are all the structures on a same level?

  As we shall see, the solutions can be divided in two main categories with respect to how they manage overlapping of elements. Firstly, there are the solutions that build the documents around a main or master structure with all the remaining structures being defined in relation to it. Secondly, there are also solutions that give no privilege to any structure. For the later, as far as the model is concerned, all the structures are equal. As we have tried to show in the introduction, non-trivial use of multi-structured documents tend to call for such an *a priori* equality of structures. Thus, we favor this last category of solutions.

- *Queries*: Does the solution come with adapted query mechanisms?

  With no doubt, the possibility to make complex queries is one of the most useful features of electronic documents. Multi-structured documents, particularly when they cause some overlapping of structure elements, often ask for subtle queries that couldn't be performed with the existing query engines. Thus, good solutions will come with new query functions and take into account the overlapping situations.

- *Updates*: Once a multi-structured document exists, how can it be modified?

  Because of the complexity of managing multiple structures for a same document, some solutions will not provide any easy way to modify an existing multi-structured document. Due to the kind of use cases aforementioned in the introduction, we give great importance to the possibility of building documents and so we rank higher the solutions allowing the modification of existing multi-structured documents.

- *Performance / Implementation*: Is there an implementation? How efficient is it?

  Finally, due to the very practical aspect of multi-structured documents and the urgent need to provide effective solutions for their management, we will consider the performance issue of the existing implementations.

2.3 SYNOPTIC VIEW OF THE SOLUTIONS

The Table 1 is a summary of the works being evaluated in the following parts of this survey section.

TABLE 1 SUMMARY OF THE EVALUATION OF RELATED WORKS

Solution		Data model	XML-compatibility	Master structure	Queries	Updates	Implementation
CONCUR		Syntactical only solution, no model.	SGML option.	The structures are independent.	No	Easy, since there is only one file.	None
TEI guidelines	redundancy	No model is provided. These are syntactical solutions.	XPath and XQuery can't be used	No	Standard XML tools are not usable	No!	No needs. They are syntactical only solutions.
	Empty elem.			Yes		Not too difficult but no tools exist for automating this process.	
	Virtual elem.						
	Stand-off markup		No generic parsing tool implements XInclude with third-party XPointers schemes. Therefore, this solution is not fully usable.		Only in theory	Only in theory	No suitable implementation of XInclude
MECS / TexMECS		A new language with a grammar that allows overlaps	No. A hierarchical TexMECS document is isomorphic to an XML document.	No. The structures are all equal.	No	Yes, since only one document.	Parsers have been implemented.

LMNL	A new language based on a notion of layers with overlapping and recursive annotations.	XML syntax is provided (it relies on empty elements). The use of standard XML tools becomes difficult.	No master structure.	No specialized query mechanism.	Yes, since only one document.	A toy parser has been implemented and some XSLT style sheets to deal with the XML syntax.
MuLaX	Modification of the XML core model	An mlx document is not compatible with XML, but there can be a projection of each structure into an XML document.	As for CONCUR, the structures are independent.	No	Yes, since only one document.	Yes + there is an editor for mlx documents written as an extension for the Eclipse IDE
Annotations Graphs	Native graph model	No	No master structure but a decomposition of the content in elementary fragments.	XPath extensions have been proposed for the XML serialization of Annotation Graphs.	Since structures and data share a same graph, updates are easy to perform.	The implementation focuses on linguistic applications
RDFTef	RDF graph model	The standard XML tools can't be used.	No master structure.	Specialized SPARQL functions can manage overlapping.	Yes, same reason as above.	Only a "toy" experiment.

EARMARK	RDF graph model. It is more of a meta-model in which most of the models in this table could be expressed.	No	No master structure.	SPARQL can be used quite straightforwardly.	Yes, same as above.	Standard Semantic Web tools are used for the implementation (Pellet, etc.)
GODDAG	Same as above.	Yes, by way of a modification of the DOM.	Same as above.	By way of new XPath axes.	Updates are not considered but individual structures can be extracted / integrated from / to a GODDAG.	Same as above.
MCT	Same as above.	Yes, at the cost of an extension to the core model of XQuery.	Yes, there is one master structure around which the others are defined.	Yes: with a new XPath step one can choose the color for an XML node.	It *might* be possible to use XUpdate with the new XPath step.	An extension to the Timber XML DB.
Delay Nodes	Add a new kind of node to the core XML model.	Provided the extension of the model, delay nodes documents can be processed by standard XML tools.	Same as above.	XPath and XQuery can be used with some restrictions.	No, updating a delay nodes document is very difficult.	As far as we know, no implementation seems to be actually available on the Web.
MSXD	Same as above.	Yes, with some XQuery functions.	No, all structures are equal.	By way of new XQuery functions.	Updates are not considered.	A prototype has been implemented.

MonetDB	Multiple rooted trees.	Yes, since it is an extension of an XML DB.	All structures are considered equals.	Four new optimized XPath axes have been added.	Updates are difficult. No dedicated tool provided.	Very optimized implementation in the context of an existing XML DB (development has been frozen in march 2011).
MSDM / MultiX	Same as above.	Yes and it only needs some XQuery functions.	Same as above.	Thanks to a few XQuery functions.	Yes, thanks to a specialized parser.	The MXQ library of XQuery functions is available on the Web.

2.4 HISTORICAL WORKS

2.4.1 CONCUR

CONCUR (ISO 8879, 1986) is an optional SGML feature with which one can define several parallel DTD for the same content. In such an SGML document, all structures share a single file. For each structure, the annotations are decorated with a specific prefix identifying a given DTD (see Listing 1 for an illustration of CONCUR with the manuscript example introduced above).

```
<!DOCTYPE S1 SYSTEM "dtds/physical.dtd">

<!DOCTYPE S2 SYSTEM "dtds/syntaxical.dtd">

<(S1)Lines>

<(S2)Words>

<(S1)Line><(S2)Word>hu</(S2)Word> <(S2)Word>þu</(S2)Word> <(S2)Word>me</(S2)Word>
<(S2)Word>hæfst</(S2)Word> <(S2)Word>afrefredne</(S2)Word> <(S2)Word>æg</(S1)Line>

<(S1)Line>þer</(S2)Word> <(S2)Word>ge</(S2)Word> <(S2)Word>mid</(S2)Word>
<(S2)Word>þinre</(S2)Word> <(S2)Word>smealican</(S2)Word> <(S2)Word>spræ</(S1)Line>

<(S1)Line>ce</(S2)Word>, <(S2)Word>ge</(S2)Word> <(S2)Word>mid</(S2)Word>
<(S2)Word>þinre</(S2)Word> <(S2)Word>wynsumnesse</(S2)Word> <(S2)Word>þines</(S2)Word></(S1)Line>

</(S2)Words>

</(S1)Lines>
```

LISTING 1 CONCUR ILLUSTRATION

Meanwhile, relationships can't be established between the different structures. Moreover, as noted by (Hilbert, Witt, & Schonefeld, 2005), when two elements of different DTD are qualifying exactly the same region, the order of their "start" and "end" tags is irrelevant. Obviously, as an SGML option, CONCUR is not compatible with XML. Though, as we should see, there have been initiatives to adapt it to the XML world. We are not aware of any query mechanism for CONCUR. The

document being unique, structures and data changes are easy. Finally, this solution has rarely been implemented and even Charles Goldfarb (as recalled in (Hilbert, Witt, & Schonefeld, 2005)), the main developer of the SGML standard, did not recommend its use.

2.4.2 TEI

The XML standard, although based on SGML, did not retained nor adapted the CONCUR modeling of multi-structured documents. However the needs were more than ever present. Thus, the XML version of the TEI (Text Encoding Initiative) guidelines offers several methods for the encoding of multiple hierarchies (TEI Consortium, 2011). These methods can be classified in four different categories : redundant encoding of information in multiple forms (C1) ; empty elements to delimit the boundaries of non-nesting structures (C2) ; breaking non-nesting elements into smaller elements linked by a 'reference' attribute (C3) ; stand-off markup approach that separates the content and the elements describing it (C4). These solutions are more like recipes than well-defined models.

With the first three categories of solutions, the standard XML tools (XPath, XQuery …) can't be used straightforwardly. It quickly becomes very difficult to express even simple queries. For the fourth solution, the guidelines advise us to use the XInclude[2] standard to establish the links between structures and content. However, the TEI also uses third-party XPointers schemes, and no generic parsing tools implement XInclude with support for such schemes (Bański, 2010).

For redundant encoding of information, there is obviously no master structure. For empty elements, and fragmentation, the elements of encoding that do not make use of the recipes can be said to belong to a master structure, and only this structure will be easy to work with (query, etc.). For stand-off markup, the structures will make reference to a decomposition of the content in elementary fragments and no master structure is as such required.

Moreover, since XPath and XQuery can't be used in a standard way, queries are either difficult or even impossible.

Updating facilities vary. For the first category (redundant encoding), updates are of course very costly. For categories C2 and C3, since the representations of the structures share a same file, updates are easy. For stand-off markup, no mechanism is provided to ease the updating process.

Finally, those solutions only consist of recipes and no implementation *per se* is provided. However, numerous projects seem to follow these recommendations for the representation of multi-structured documents when overlapping occurs ony rarely.

2.5 FORMAL MODELS

2.5.1 TEXMECS

To make up for the limits of existing markup languages, other studies have been carried out in order to define new syntaxes. MECS (Multi-Element Code System) made the overlapping of elements possible. TexMECS (Huitfeldt & Sperberg-McQueen, 2003) is a follow-up of MECS where complex structures can be defined with elements having many parents (see Listing 2 for an illustration of a TexMECS encoding of the manuscript example introduced above).

---

[2] www.w3.org/TR/xinclude/

```
<lines|<words|
<line|<word|hu|word> <word|þu|word> <word|me|word> <word|hæfst|word>
<word|afrefredne|word> <word|æg|line><line|þer|word> <word|ge|word>
<word|mid|word> <word|þinre|word>
<word|smealican|word><word|spræ|line><line|ce|word>, <word|ge|word>
<word|mid|word><word|þinre|word> <word|wynsumnesse|word>
<word|þines|word>|line>
|words>|lines>
```

LISTING 2 TEXMECS ILLUSTRATION

In general, TexMECS documents are not isomorphic to XML documents and so the XML tools cannot be used. However, for documents with no overlaps, a direct translation to XML is feasible. As far as we know, no query mechanisms have been developed. Moreover, all the structures and the primary data sharing a same representation (i.e. a file), it remains easy to update them. Finally, a TexMECS well-formed-ness checker has been implemented[3].

2.5.2 LMNL

LMNL (Layered Markup and aNnotation Language) (Tennison & Piez, 2002) defines a specific syntax based on the notion of range, allowing the encoding of multiple structures where elements can overlap (see Listing 3 for an illustration). However, LMNL is first of all a data model and not only a syntactical representation. A LMNL document is based on a *text layer* made of a sequence of atoms (for example: Unicode characters). *Ranges* are used to define fragments of content. Ranges can be annotated with new text layers that themselves can have ranges…

```
[Manuscript [Title}Manuscript 1{title]}
  [Lines}[Words}
  [Line}[Word}hu{Word]… …
  [Word}hæfst{Word] [Word}afrefredne{Word] [Word}æg{Line]
  [Line}þer{Word]… …
{Words]{Lines]
{Manuscript]
```

LISTING 3 LMNL ILLUSTRATION

Pathways exist to convert to and from XML documents by way of milestones (i.e. empty elements). Obviously, when overlapping happens, it then becomes difficult to use standard XML tools. As far as we know, no query mechanisms have been developed. Moreover, all the structures and the primary data sharing a same file, it remains easy to update them. Finally, XSLT stylesheets have been developed to deal with the XML representation of a LMNL document[4].

2.5.3 MuLaX

MuLaX (Hilbert, Witt, & Schonefeld, 2005) is an adaptation of the SGML CONCUR option to the XML formalism. It is intended as a document format that can merge in a uniform way a collection of several XML documents into one document. A layer ID prefixes each tag name. A MuLaX document can be projected onto its initial XML documents (one XML

[3] http://decentius.aksis.uib.no/servlets/mlcd/web/WFCheck.jsp

[4] http://www.piez.org/wendell/LMNL/lmnl-page.html

document for each layer). Each projection results in a well-formed XML document. Listing 4 is an example of the MuLaX syntax applied to our running example.

```
<?mlx version="1.0" encoding="iso-8859-1">
<(1)Lines><(2)Words>
<(1)Line><(2)Word>hu</(2)Word> <(2)Word>þu</(2)Word> <(2)Word>me</(2)Word>
<(2)Word>hæfst</(2)Word> <(2)Word>afrefredne</(2)Word> <(2)Word>æg</(1)Line>
<(1)Line>þer</(2)Word> <(2)Word>ge</(2)Word> <(2)Word>mid</(2)Word>
<(2)Word>þinre</(2)Word> <(2)Word>smealican</(2)Word> <(2)Word>spræ</(1)Line>
<(1)Line>ce</(2)Word>, <(2)Word>ge</(2)Word> <(2)Word>mid</(2)Word>
<(2)Word>þinre</(2)Word> <(2)Word>wynsumnesse</(2)Word>
<(2)Word>þines</(2)Word>
</(1)Line>
</(2)Words></(1)Lines>
```

LISTING 4 MULAX ILLUSTRATION

Standard XML tools can only be used on the individual XML projections. Thus, inter-structural relationships remain out of reach. No query operators are defined but the authors give some hints on how XPath could be extended to navigate in a MuLaX document. Moreover, updates are difficult because working on MuLaX documents implies frequent projections to XML files. Indeed one should never modify a projection but only the main document. Finally, an editor has been developed as an Eclipse plugin for the creation of MuLaX documents.

2.5.4 ANNOTATIONS GRAPH

Annotation graphs (Bird, Liberman, & Walker, 1999) appeared in the context of language analysis. Thus, they take into account linguistic domains (phonetics, prosody, morphology, syntax, etc). In our context, these domains can be considered as multiple structures. The same text fragment could be annotated by edges belonging to different structures (as shown in Figure 4). The data fragmentation corresponds to words or to characters if it is necessary. The labels associated with the oriented edges identify elements of annotation within a specific structure (for example 'S2:w').

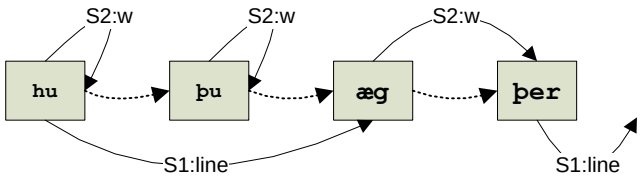

FIGURE 4 ANNOTATIONS GRAPH ILLUSTRATION

The physical model could be serialized to XML (using attributes of types ID and IDREF). Thus available XML tools and languages can be used for each extracted hierarchical structure. However, the document as a whole cannot be the object of any treatment by standard XML tools. A prototypical query language (Bird, Buneman, & Tan, Towards a Query Language for Annotation Graphs, 2000) has been defined for annotations graphs. Moreover, structures remain easy to update since they share the same data. Finally, there exists an implementation called AGTK (Annotation Graph Toolkit)[5].

[5] http://agtk.sourceforge.net/

2.5.5 RDFTef

In a very similar way to the Annotation Graph proposal, RDFTef (Tummarello, Morbidoni, & Pierazzo, 2005) makes use of the RDF (Resource Description Framework) formalism to provide a framework for the modeling of multi-structured documents. This method takes advantage of the RDF graph model, which may be used to encode complex structures with overlapping elements. However, contrary to the Annotation Graphs, RDFTef is based on a well-defined standard formalism. Moreover, this RDF based proposal is not oriented toward a specific domain. RDF can be serialized to XML but since the model is a graph the use of standard XML tools is not satisfactory. However, standard RDF query tools (such as SPARQL) can be used but medium complexity queries can be difficult to formulate. Moreover, structures and data changes are theoretically possible since there is only one graph. Finally, a prototype implementation based on the Jena Java RDF library is available online[6].

2.5.6 EARMARK

Among the RDF based solutions, EARMARK (Peroni & Vitali, 2009) is certainly the most convincing one. The notions of "location", "range", "markup item", etc. used for the modeling of multi-structured documents are precisely defined within an OWL ontology (see Figure 5 for an excerpt of the ontology).

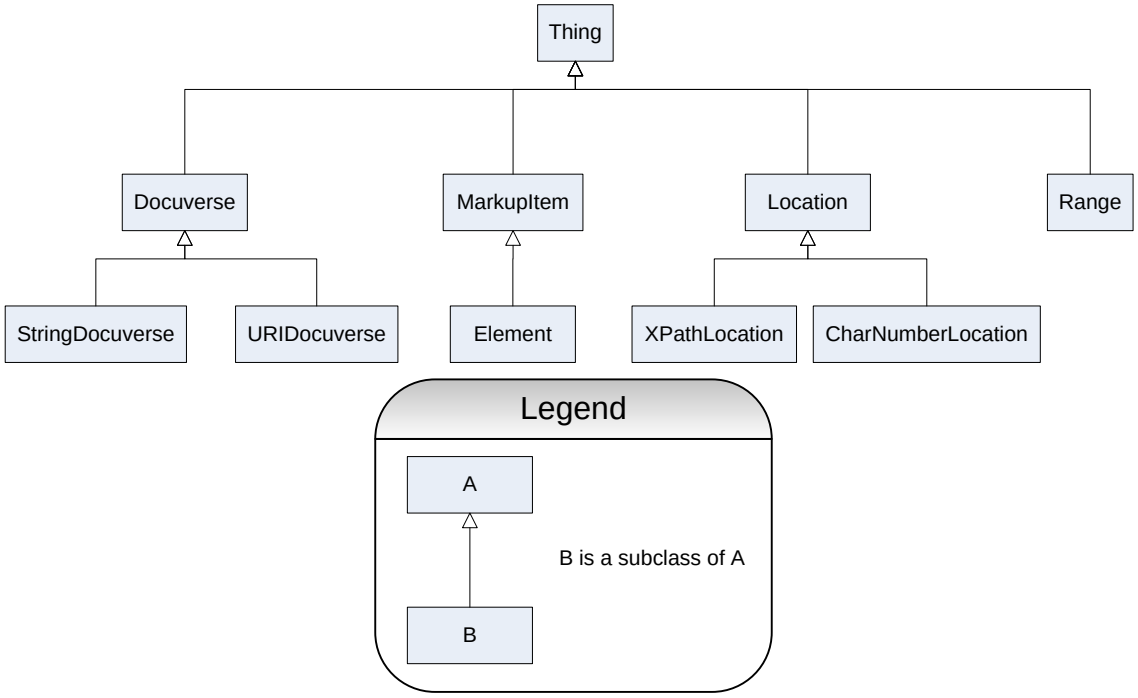

FIGURE 5 EXCERPT FROM THE EARMARK ONTOLOGY

Tools are provided to convert an EARMARK instance into an XML document with, for example, empty elements. However, standard XML tools can't be used directly with an EARMARK document. Being a graph model, EARMARK doesn't make any low-level distinction between structures. Moreover, the SPARQL language can be used for querying the documents. For updates, the graph can be easily manipulated through standard RDF APIs. Finally, a prototype implementation exists, though it doesn't seem to be available on the Web. However, the main contribution lies in the OWL ontology and can be implemented with any standard OWL engine.

[6] http://rdftef.sourceforge.net/

2.5.7 GODDAG

The GODDAG (General Ordered Descendant Directed Acyclic Graph) (Dekhtyar & Iacob, 2005) logical model can be used for the creation of an internal representation of concurrent XML structures. The obtained graph structure resolves the overlapping problem. As shown on Figure 6, several trees are defined over the same content by sharing their leaves (for example: textual fragments). All nodes have at least one common ancestor: the document root.

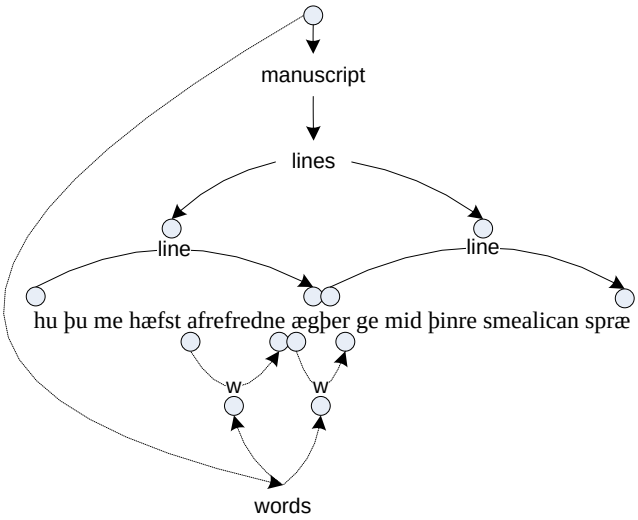

FIGURE 6 ILLUSTRATION OF THE GODDAG MODEL

First of all, functions for importing (resp. exporting) from (resp. to) XML are defined. Moreover, new axes are provided for the XPath language. Therefore, the standard XML tools can be used in a natural way. For querying GODDAG, the authors propose an extension of the XPath language. This extension is based on a set of new axes that lead to nodes according to their relations and independently from the structures to which they belong: it is possible to reach every ancestor (axis *xancestor*) of a node in every hierarchy to which it belongs. In a similar way, other axes are defined: *xdescendant*, *xfollowing*, *xpreceding*, *overlapping*, *following-overlapping* and *preceding-overlapping*. Moreover, operators are defined to manage changes in the structures. However, the raw data cannot be updated. Finally, the implementation is based on a generalization of the DOM tree for the representation of multi hierarchical XML documents.

2.5.8 MCT

The MCT model (Multi-Colored Trees) (Jagadish, Lakshmanan, Scannapieco, Srivastava, & Wiwatwattana, 2004) is an extension of the XML model making possible the representation of multiple trees that share a same content. It relies on the tree coloring technique. A color is associated with each tree. A node may have multiple colors: the color of the main tree to which it belongs and colors for other trees. Figure 7 illustrates this model with our running example of a manuscript transcription. We see that three of the unit nodes share two colors: one for the syntactical structure of words and another one for the physical structure of lines.

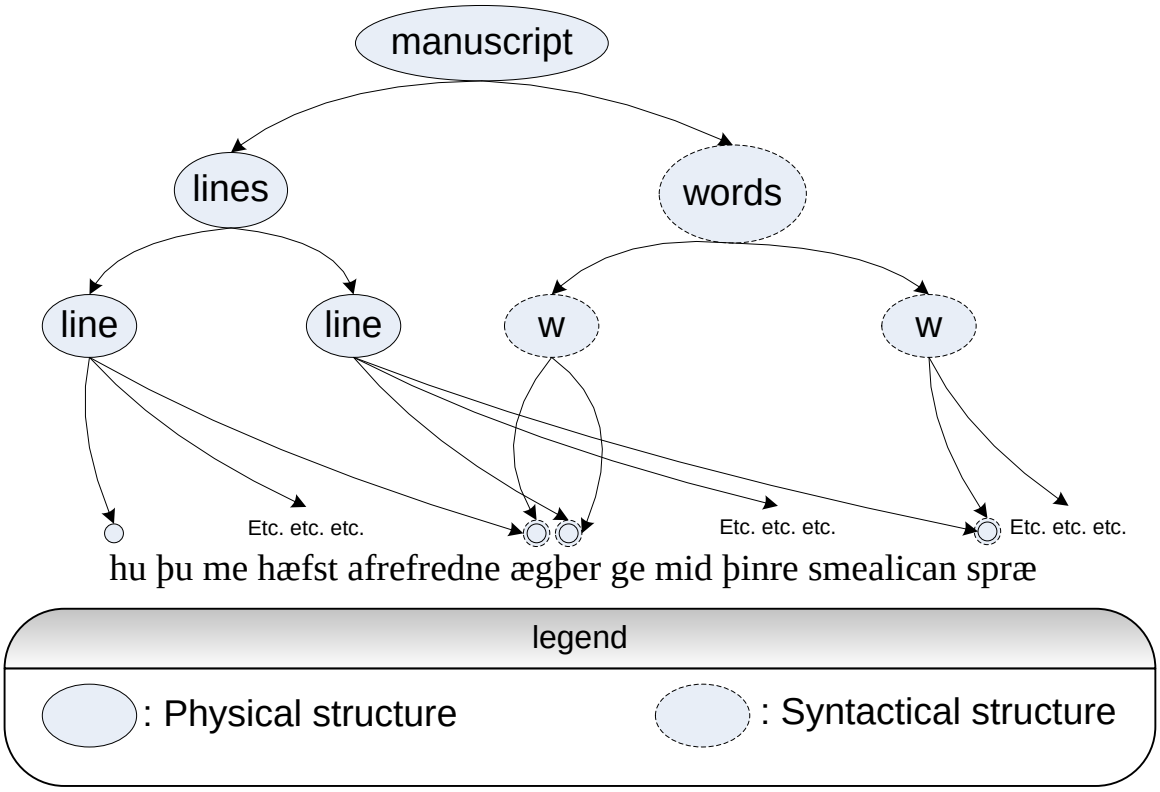

FIGURE 7 ILLUSTRATION OF THE MCT MODEL

Navigation inside the multi-hierarchy is possible by means of the multicolored nodes. Indeed, the XPath step has been extended with color choice. Thus, the model can be manipulated with standard XML tools. However, one has to choose a first color. Then the new structures are built relatively to this first structure which becomes a master structure. Together with the new XPath step, an XQuery extension has been proposed for, among others features, creating new colored nodes. Moreover, concerning updates, the authors believe that, through the XPath extension they offer, the XUpdate language could be adapted. Finally, the model has been implemented as an extension to the Timber[7] XML database.

2.5.9 DELAY NODES

The XDM[8] data model on which are based XPath and XQuery, distinguishes seven kinds of nodes in a document tree: document, element, attribute, text, namespace, processing instruction, and comment. To address the problem of concurrent hierarchy, Jacques Le Maître in (Le Maître, 2006) adds a new kind of node: the delay node. A delay node is the virtual representation, by an XQuery query, of some of the descendant of one of its ancestors as shown in Figure 8.

---

[7] http://www.eecs.umich.edu/db/timber/

[8] http://www.w3.org/TR/xpath-datamodel/

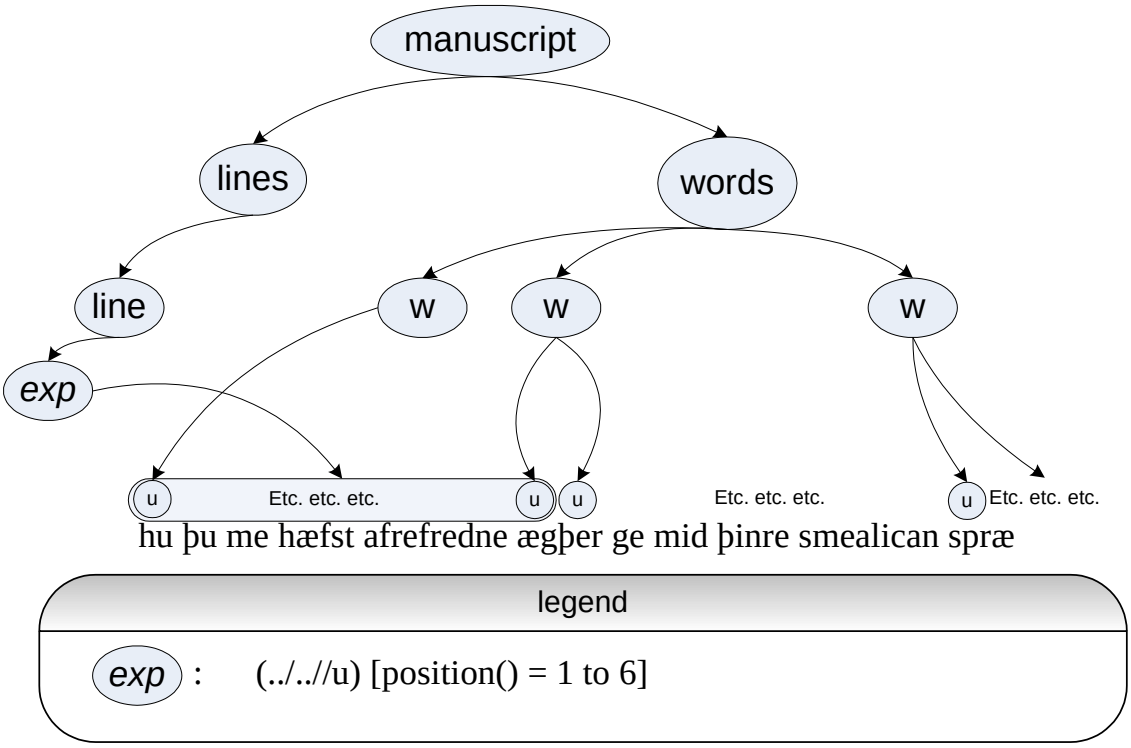

FIGURE 8 ILLUSTRATION OF THE DELAY NODES ("EXP" REPRESENTS THE DELAY NODE)

After the XDM model has been modified by the addition of delay nodes, no XPath nor XQuery extensions are needed. However the ability to navigate among the concurrent trees is only valid for the descending axis: child and descendant. Going upward from a delay node is not possible. Moreover, one structure is considered as the main structure, while others make use of delay nodes. Except for the inability to navigate inside concurrent trees differently than along the descending axis, XQuery can be used quite straightforwardly. However, changes in data or structures are very difficult to achieve since we should modify the queries of the delay nodes at a syntactical level. Finally, in (Le Maître, 2006), it is said that a prototype has been implemented on top of the authors' own XQuery engine. However, it doesn't seem to be available on the Web.

2.5.10 MSXD

MSXD (Multi-Structured XML Documents) (Bruno & Murisasco, 2006) is a proposal that provides a formal model and a query language defined as an extension of XQuery that let the structure of an XQuery unchanged. Moreover, users can annotate structural elements. MSXD is also one of the first attempts to define a schema for multi-structured textual documents. However, from a practical point of view, the need to define (a great number of) constrains between structures is not obvious. The core model of MSXD is based on the use of hedges (the foundation of RelaxNG). In fact, each structure must be hierarchical and is associated with a RelaxNG Schema. Then, Allen's relations (after, equals, overlaps, etc.) can be defined between fragments of different structures. Thus, the individual structures are classic XML documents. Moreover, syntax has been defined to represent the union of the multiple structures in a single XML document. XQuery functions are provided in order to deal with multi-structured documents (the core of the XPath/XQuery model doesn't need to be modified). However, modifications of structures and data remain impossible. Finally, a prototype implementation is available online[9] with both the MSXD model and the new XQuery functions.

[9] http://sis.univ-tln.fr/msxd/

2.5.11 MONETDB / XQUERY

MonetDB/XQuery is an XML DBMS. MonetDB/XQuery stand-off extension (Alink, Bhoedjang, Vries, & Boncz, 2006) is an efficient implementation of query operators for multi-structured documents represented by stand-off markup. Stand-off annotations are modelled with a subset of the interval algebra. The annotated binary object (text, video, etc.) must be addressable with intervals. An interval is a couple *(start,end)* where *start* and *end* positions must be of the same data type and this data type must support full ordering (for example integers). Since the solution supports the annotation of non-contiguous regions, the interval algebra is reduced to the two relations of *containment* and *overlap*. See Listing 5 for an illustration of the MonetDB stand-off extension syntax.

```
<?xml version ="1.0" encoding ="utf -8"? >
<manuscript>
  <physical>
    <lines>
      <line start=”1” end=”28”/>
      <line>
        <region>
          <start>29</start>
          <end>59</end>
        </region>
      </line>
      <!-- Etc. etc. -->
    </lines>
  </physical>
  <syntactical>
    <words>
      <word start=”1” end=”2”/>
      <!-- Etc. etc. -->
      <word start=”27” end=”31”/>
      <!-- Etc. etc. -->
    </words>
  </syntactical>
</manuscript>
```

LISTING 5 ILLUSTRATION OF MONETDB / XQUERY SYNTAX

With the provided XPath steps and the integration to an existing XML database management system; this proposal can be used with the standard XML tools. Moreover, this solution offers an efficient implementation of stand-off joins as new XPath steps. Thus, the multi-structured documents can be queried quite naturally. However, as for other stand-off solutions, updates can be difficult, no specific tools being provided. Finally, this solution relies on MonetDB, a well-known, efficient and open-source DBMS[10]. However, the development of MonetDB/XQuery has been frozen in March 2011[11] and the project will not be ported to MonetDB version 5.

[10] http://monetdb.cwi.nl/

[11] http://www.monetdb.org/XQuery

2.6 CONCLUSION

Finally, from the set of dimensions we chose to keep and the requirements we expressed for each one of them (see paragraph 2.2 p. 6), we can now explain what makes our own model necessary. We should sum up some important points of the previous solutions.

The TEI solutions are not based on a proper formal model and can't be used with standard XML tools. CONCUR was certainly interesting but came with no tools for dealing with multi-structured documents.

MuLaX transposed the CONCUR idea to the XML world but, since there is no convenient way for querying MuLaX document, this solution can't be retained.

TexMECS and LMNL chose to develop new models, grammars and syntaxes to answer the needs of multi-structured documents management; however with their current limitations they cannot be considered as generic solutions.

Annotation Graphs, RDFTef and then EARMARK avoid the problems caused by the tree model of XML by considering formalisms and models that work directly with graphs. The choice of RDF with SPARQL, its query language, makes a well-adapted solution. However, we strongly believe that the XML-compatibility criterion is essential. Indeed, a large number of the existing projects dealing with complex electronic documents are using XML. Moreover, standardization initiatives, such as the TEI or EAD (Encoded Archival Description)[12] etc. are also using XML.

Among the stand-off markup solutions, Delay Nodes and MCT are very clever ones. Their models are interesting. However, they have to give privileges to one of the structures; and it has been explained, in paragraph 1.2.5 page 5, that a free fragmentation of the raw content and truly independent structures are essential characteristics for a satisfactory multi-structured documents management system.

MonetDB and MSXD are based on the GODDAG stand-off markup solution but none of them can easily manage the modifications of data and structures (this is common to many stand-off techniques). Moreover, MonetDB and MSXD both rely on extensions of XPath with new steps. This choice tends to make the solutions dependent on a particular implementation.

Thus, we now propose our own model: MSDM, a Multi-Structured Document Model. It is based on a stand-off markup approach and thus relies on a well-defined formal model of multiple rooted trees. Being implemented through a few XQuery functions, it remains fully compatible with the galaxy of known XML tools. It doesn't rely on any privileged master structure and is therefore well adapted for the modeling of complex multi-structured documents. Finally, updating is possible thanks to a specialized parser. This model will now be described in full details.

3 MSDM: MULTI-STRUCTURED DOCUMENT MODEL

To answer the problem of multiple structuring, we have proposed a specific model called the Multi-Structure Document Model (MSDM) (Chatti, Kaouk, Calabretto, & Pinon, 2007). As many of the previous solutions, ours is based on the stand-off markup method where structures point to the content stored separately. However, we approach the problem in a more general way. In fact, we assume that multiple structures can share the exact same content fragments, and not necessarily exactly the same content. Thus, for our model, concurrent structures are a particular case of multi-structured documents.

[12] http://www.loc.gov/ead/

In this model, which is inspired by the model defined in (Abascal, et al., 2003), a multi-structured document is defined using the three following notions:

- *Documentary Structure (DS)*: this is a description of a document content defined for a specific use. Such a structure may be, for example, a physical structure used for presentation purposes, or a syntactical structure defined for statistical analysis, etc.
- *Basic Structure (BS)*: this structure organizes the content in disjoint elementary fragments. These fragments are then composed in order for the documentary structures to refer to their content by pointing to these composition nodes.
- *Correspondence*s: a correspondence is a relationship between two elements of two distinct structures. The source of a correspondence is always an element of a documentary structure. When the target is a composition node of the basic structure, the correspondence is used for identifying the content of a documentary structure node. Otherwise, it indicates a relation between two documentary structures, the nature of which depends on the context of the application, it could, for example, indicates a synonymy between two nodes of two distinct DS, it could also, as in our manuscript example, link the transcription of a line to the corresponding region of the manuscript's image, etc.

Figure 9 illustrates these notions on the previous example of a manuscript transcription.

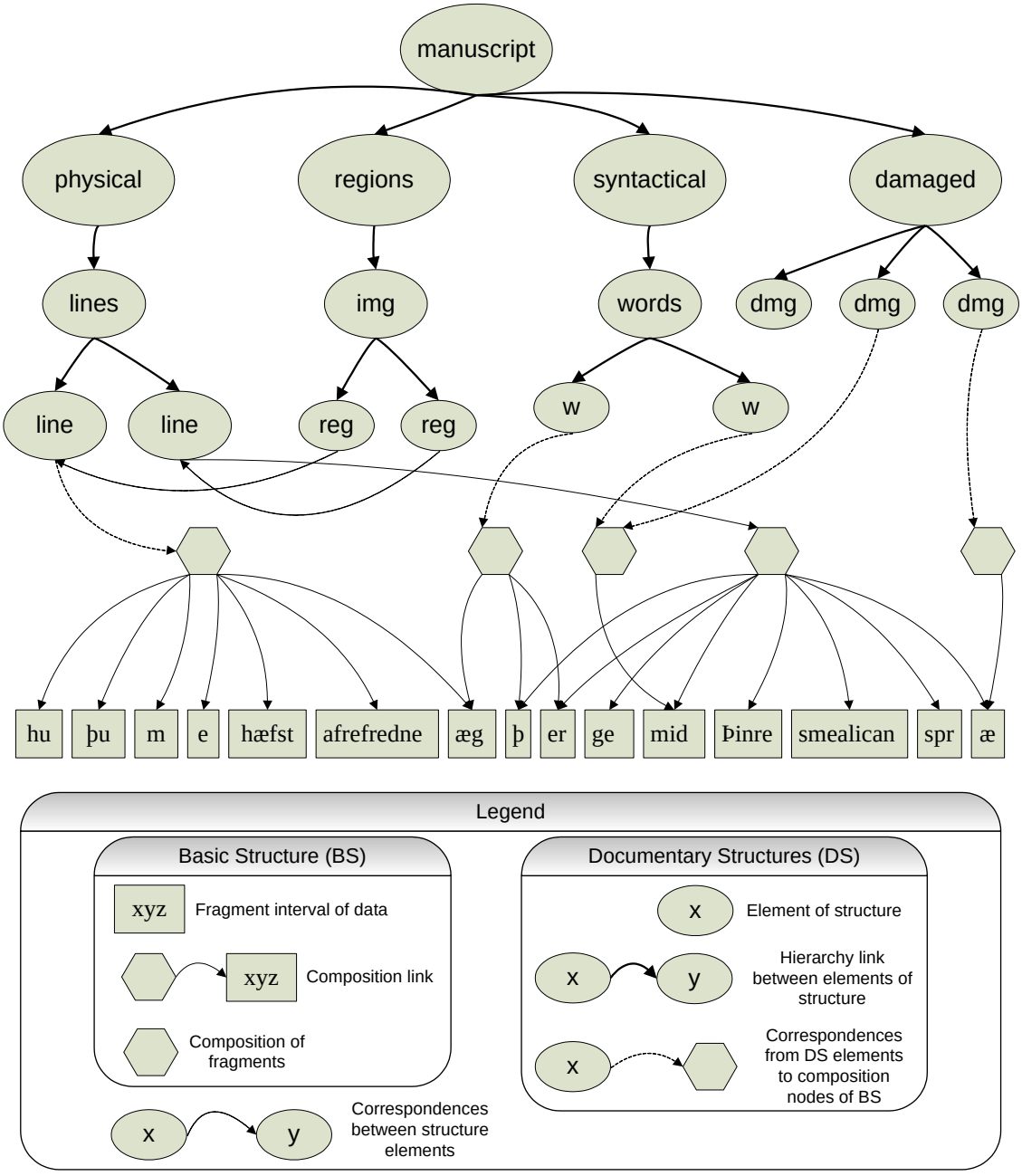

FIGURE 9 ILLUSTRATION OF THE MULTI-STRUCTURED DOCUMENT MODEL (MSDM)

To clarify even further the model, Listing 6 gives an XML representation of the four DS of Figure 9 before factorization of the content inside BS.

```
<physical>
  <lines>
    <line n="1">hu þu me hæfst afrefredne æg</line>
    <line n="2">þer ge mid þinre smealican spræ</line>
    <line n="3">ce, ge mid þinre wynsumnesse þines</line>
  </lines>
</physical>

<syntactical>
  <words>
    <w>hu</w> <w>þu</w> <w>me</w> <w>hæfst</w> <w>afrefredne</w> <w>ægþer</w>
<w>ge</w> <w>mid</w> <w>þinre</w> <w>smealican</w> <w>spræce</w> <w>ge</w>
       <w>mid</w> <w>þinre</w> <w>wynsumnesse</w> <w>þines</w>
  </words>
</syntactical>

<damaged>
  <res>þu m</res> <dmg>er</dmg> <dmg>mid</dmg> <dmg>æ</dmg> <dmg>g</dmg>
<dmg>þ</dmg> <dmg>re</dmg> <dmg>e</dmg> <res>s</res>
</damaged>

<text-regions>
  <image src="manuscript.png">
    <region num="reg.1" description="first line">
      <zone x1="43" y1="50" x2="460" y2="94"/>
    </region>
    <region num="reg.2" description="second line">
      <zone x1="43" y1="108" x2="486" y2="152"/>
    </region>
    <region num="reg.3" description="third line">
      <zone x1="43" y1="166" x2="536" y2="210"/>
    </region>
    <region num="reg.4" description="word on two lines, 1 and 2">
      <zone x1="410" y1="50" x2="460" y2="94"/>
      <zone x1="43" y1="108" x2="93" y2="152"/>
    </region>
    <region num="reg.5" description="word on two lines, 2 and 3">
      <zone x1="'414" y1="108" x2="486" y2="152"/>
      <zone x1="43" y1="166" x2="72" y2="210"/>
    </region>
  </image>
</text-regions>
```

LISTING 6 THE FOUR DOCUMENTARY STRUCTURES OF THE MANUSCRIPT EXAMPLE : THE PHYSICAL STRUCTURE COMPOSED OF LINES, THE SYNTACTICAL STRUCTURE COMPOSED OF WORDS ('W' IS USED FOR A WORD), THE DAMAGED STRUCTURE WITH THE RESTORED FRAGMENTS ('RES') AND THE DAMAGED FRAGMENTS ('DMG'), AND FINALLY THE TEXT-REGIONS STRUCTURE COMPOSED OF THE DESCRIPTION OF RECTANGULAR ZONES OF THE IMAGE

3.1 FORMALIZATION

We present in this section the formal description of our multi-structured document modeling approach. We first define the concept of multi-structured document and then we describe formally all the necessary notions.

Definition 1 (Multi-structured Document):

A multi-structured document D is a graph defined by the triplet $D = \langle BS, \Sigma, C \rangle$ where:

- BS is an oriented graph called the basic structure of D;
- Σ is a set of graphs called the documentary structures;
- C is a set of relations between nodes of Σ-graphs or BS. We call these relations *correspondences* between structures.

3.1.1 BASIC STRUCTURE

Notations:

- F(D) designates the set of all possible content fragments of a document D.
- $\phi(D) = \{\varphi : (F(D))^p \rightarrow F(D) \mid p \geq 1\}$ is the set of all the functions that compose a piece of content from a series of smaller pieces. For example, the function that creates the string "ab" by concatenation of the two fragments $(a,b) \in F(D)^2$ is an element of $\phi(D)$.

The compositions are possible only for fragments with a same media type: this is the *homogeneity constraint* for compositions. To ensure a minimum level of abstraction, we do not debate here the details of composition functions. However, we will later illustrate our model with an example where a specific use of fragment compositions will appear.

Definition 2 (Basic Structure):

The basic structure of a multi-structured document D is an internal structure that stands as a base for sharing the same content among several structures. It is formally defined as a graph BS = (v,e) where:

- root is the root node of BS, each node is part of a path originating from it. This node has no associated semantic; it only provides connectivity to the graph.
- $F_{Base}(D) \subset F(D)$ is the set of all disjoint fragments, called base fragments, from which other fragments can be composed and associated to document structures through correspondences. Each node $n \in F_{Base}(D)$ is the destination of one or more edges from composition nodes.
- N_{comp} is the set of composition nodes, i.e. the nodes that represent parts of content composed of smaller fragments. Each node $n \in N_{comp}$ is the destination of exactly one edge with origin the root node. So the subgraph of BS whose nodes are $\{root\} \cup N_{comp}$ is a trivial tree. Moreover each node $n \in N_{comp}$ is labelled with a function $f \in \phi(D)$ that prescribes the composition of the elements of $F_{Base}(D)$ linked to n.
- $v = \{root\} \cup N_{comp} \cup F_{Base}(D)$ is the set of vertices of BS
- comp(n) with $n \in N_{Comp}$ is the set of all the edges from the composition node n to one or more elements of $F_{Base}(D)$. Each of these edges is labeled with an integer, that makes comp(n) a totally ordered set. This order may be used by a function $f \in \phi(D)$ associated with n to compose the elements of $F_{Base}(D)$ in the appropriate order.
- $e = \{(root,c) \mid c \in N_{comp}\} \cup (\cup_{n \in Ncomp} comp(n))$ is the set of edges of BS.

3.1.2 DOCUMENTARY STRUCTURES

Definition 3 (documentary Structure):

A documentary structure is a set of structured descriptors applied to documentary content. Formally, a documentary structure is a tree defined by: DS = $\langle E, A, L, l_E, l_A \rangle$ where:

- E is a finite set of vertices we call *structural elements*.
- $A \subset E \times E$ is a finite set of binary relations between structural elements. These relations must form a tree.
- L is a finite set of labels.
- $l_E : E \rightarrow L$ is a function that associates each structural element in E with a label in L.
- $l_A : A \rightarrow L$ is a function that associates each edge in A with a label in L.

In this definition we followed a simple approach in the representation of documentary structures. The most important thing in our model is not the representation of documentary structures, but the representation of the basic structure and of the correspondences.

3.1.3 CORRESPONDENCES

The multi-structured document is a graph consisting of a set of graphs (documentary structures and a basic structure) whose elements (i.e. composition nodes of the basic structure and structural elements of the documentary structures) can be linked by correspondence relationships.

A correspondence, noted DS → BS, associates the document structures with their original contents (PCDATA for the textual content, in XML terminology) reconstructed from the basic structure. There are also DS → DS correspondences and the following MultiX formalism offers a way for representing them, however they can also be implemented by means of any XML linking mechanism and don't *per se* belong to the MSDM model.

In the next section, the MultiX formalism is used to XML-encode the multi-structured document created from these four structures.

4 MULTIX: XML APPLICATION BASED ON MSDM MODEL

The MultiX formalism is an XML application based on the MSDM model. It consists in the serialization of a multi-structured document into a well formed XML document. We call a multi-structured document encoded within this formalism: a MultiX document. Such a document is composed of three parts: the documentary structures (DS), the basic structure (BS) and the correspondences. A MultiX document must have the skeleton depicted on Figure 10.

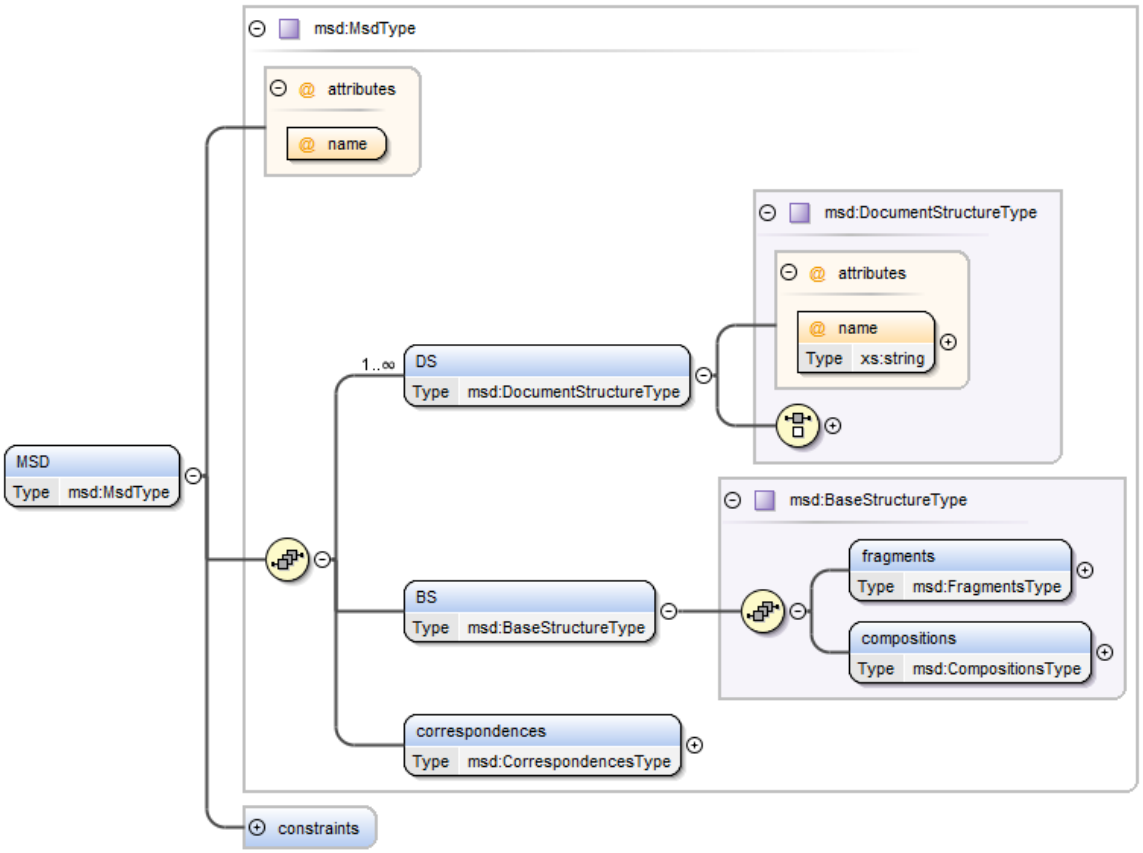

FIGURE 10 SKELETON FOR A MULTIX DOCUMENT

4.1 ENCODING THE BASIC STRUCTURE

The basic structure provides an organization of the content into a set of disjoint fragments and a set of fragments' compositions from which PCDATAs of the original documentary structures can be rebuilt. The basic structure is mainly defined by the fragmentation of shared content so as to avoid content redundancy. For example, if we factorize the content of the first line in the physical structure with the first six words in the lexical structure and the first element in the damaged structure, we obtain this set of fragments: {"hu", "þu", "m", "e", "hæfst", "afrefredne", "æg"}. This is the minimal set of disjoined fragments that can be used to rebuild the three documentary structures. In case of our entire running example, the basic structure's fragments would be encoded in the following way:

```
<msd:BS>
    <msd:fragments>
          <msd:frag id="SP"> </msd:frag>
          <msd:frag id="F1">hu </msd:frag>
          <msd:frag id="F2">&#254;u</msd:frag>
          <msd:frag id="F3">m</msd:frag>
          <msd:frag id="F4">e</msd:frag>
          <msd:frag id="F5">h&#230;fst</msd:frag>
          <msd:frag id="F6">afrefredne</msd:frag>
          <msd:frag id="F7">&#230;g</msd:frag>
          <msd:frag id="F8">&#254;</msd:frag>
          <msd:frag id="F9">er</msd:frag>
```

```
        <msd:frag id="F10">ge</msd:frag>
        <msd:frag id="F11">mid</msd:frag>
        <msd:frag id="F12">&#254;inre</msd:frag>
        <msd:frag id="F13">smealican</msd:frag>
        <msd:frag id="F14">spr</msd:frag>
        <msd:frag id="F15">&#230;</msd:frag>
        <msd:frag id="F16">ce</msd:frag>
        <msd:frag id="F17">g</msd:frag>
        <msd:frag id="F18">e</msd:frag>
        <msd:frag id="F19">mid</msd:frag>
        <msd:frag id="F20">&#254;</msd:frag>
        <msd:frag id="F21">in</msd:frag>
        <msd:frag id="F22">re</msd:frag>
        <msd:frag id="F23">wynsumnesse</msd:frag>
        <msd:frag id="F24">&#254;in</msd:frag>
        <msd:frag id="F25">e</msd:frag>
        <msd:frag id="F26">s</msd:frag>
        <msd:frag id="F27">, </msd:frag>
    </msd:fragments>
    <msd:compositions>><!-- compositions --></msd:compositions>
```

Apart from fragments, the basic structure is also made of composition nodes (see Figure 11) which define compositions of fragments later associated to documentary structures through correspondences.

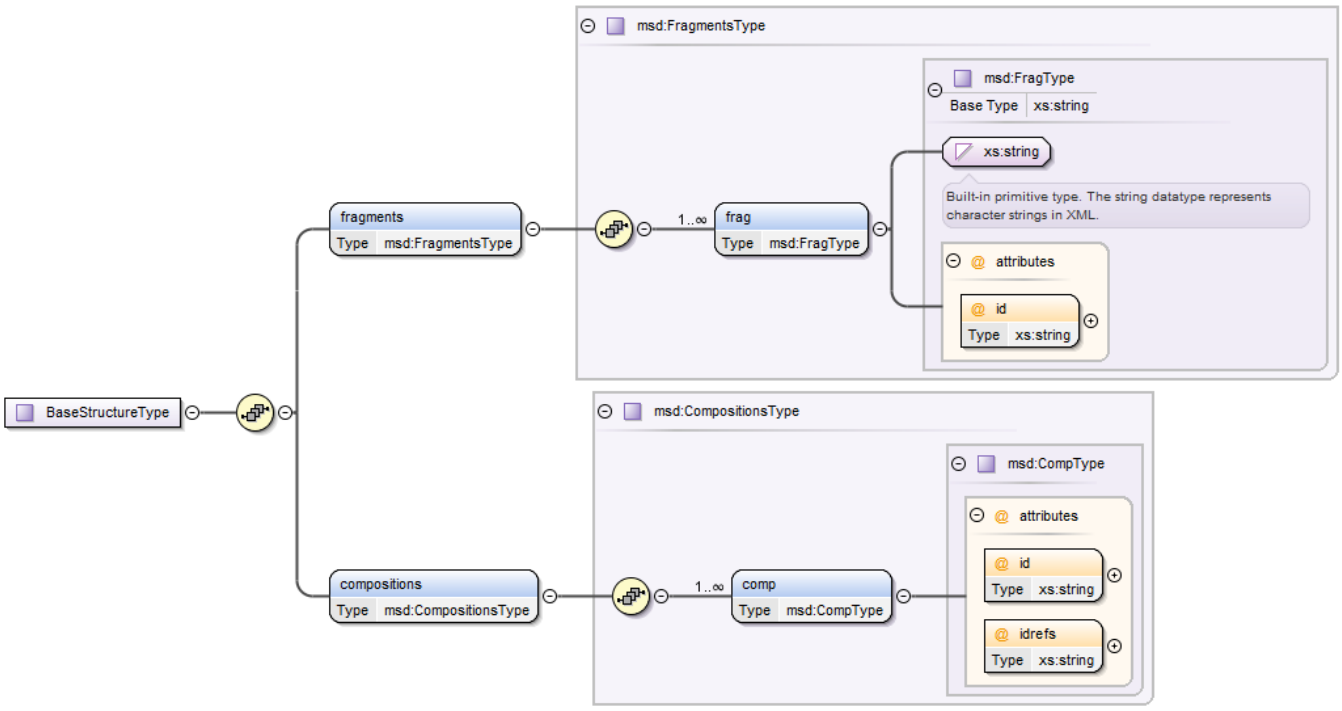

FIGURE 11 SKELETON FOR THE BASIC STRUCTURE

For example, the composition enabling the reconstitution of the content of the first line element in the physical structure will be encoded as follows:

```
<msd:comp id="CL1" idrefs="F1 SP F2 SP F3 F4 SP F5 SP F6 SP F7"/>
```

The ‘id’ attribute identifies the composition element. The ‘idrefs’ attribute value is a sequence of fragment identifiers. Each composition provides a string obtained by concatenating the identified content fragments. The ‘SP’ fragment must always be available and provides a single space character.

The ‘msd:compositions’ element of the basic structure must contain all the compositions necessary for rebuilding the documentary structures content.

4.2 ENCODING OF DOCUMENTARY STRUCTURES AND CORRESPONDENCES

A MultiX document may have several documentary structures encoded inside ‘msd:DS’ elements. A documentary structure in the MultiX formalism is similar to its standard XML representation except for the actual content that is replaced by ‘DS→BS’ correspondences pointing to composition nodes.

In general, a correspondence has a source, a target and a label. For the ‘DS→BS’ correspondences the target is always a composition in the basic structure. The source of this correspondence type represents the position in the documentary structure where the composed content should be placed.

Inside a MultiX document, the correspondences are defined in the ‘msd:correspondences’ element (see Figure 12). The source is located by the ‘msd:anchor’ element inserted at the desired position in the documentary structure.

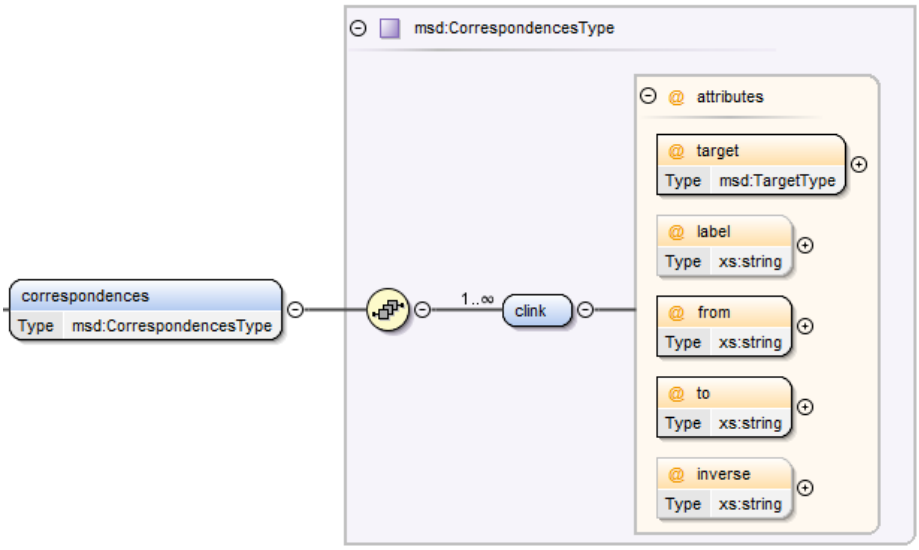

FIGURE 12 SKELETON FOR THE CORRESPONDENCES

For example, to rebuild the original PCDATA of the third line element of the physical structure, we can define two correspondences by inserting anchors in appropriate positions:

```
<line n="3"><msd:anchor id="A.lne.3.1"/><msd:anchor id="A.lne.3.2"/></line>
```

We then create the ‘msd:clink’ elements as follows:

```
<msd:correspondences>
    <!-- ... -->
```

```
    <msd:clink target="BS" label="text content" from="A.lne.3.1" to="CL3_1"/>
    <msd:clink target="BS" label="text content" from="A.lne.3.2" to="CL3_2"/>
    <!-- ... -->
</msd:correspondences>
```

The target attribute of the 'msd:clink' element indicates the type of the correspondence: "BS" for a DS→BS correspondence, and "DS" for a DS→DS correspondence. With this external form of correspondences the anchors can be reused. For example, an anchor that is a source of a DS→BS correspondence can also be used as the source or the target element of relations between documentary structures. In addition, compared with XPath expressions, the use of an anchor minimizes the required correspondences' updates after the modification of the documentary structures.

5 QUERYING MULTIX DOCUMENTS

There are several XML query languages that may be used to query MultiX documents. However, when using these languages the MultiX documents are viewed as pure XML documents. An important effort is then necessary to create queries that take into account the semantic of the MultiX formalism. To be queried easily, the MultiX formalism needs an adapted query language. XQuery (W3C, 2010) is a powerful language and a W3C recommendation inspired by SQL. In order to adapt this language to MultiX, we have implemented a library of XQuery functions (MXQ[13]) for exploiting MultiX documents. These functions aim at simplifying the writing of queries involving several structures. With them, we can for example check if some content is shared by two or more structures. Moreover, it becomes easy to explore relations between elements of different structures. Before demonstrating how to use our library through some examples, we shall describe a specific function called 'rebuild'.

5.1 DOCUMENTARY STRUCTURE RECONSTITUTION

If we apply queries on MultiX documents, items returned are incomplete or incomprehensible. Indeed, instead of textual content we get references to correspondences to the basic structure. Other elements, referring to relations between documentaries structures can also be found. Therefore it is essential to translate these references properly, and to present the results in a readable form, as if the structures were encoded separately. These transformations are performed in a transparent way thanks to the 'rebuild' function. This function has the following signature:

```
mxq:rebuild($elem-seq as element()*) as element()*
```

With this function we can rebuild a sequence of elements belonging to document structures. The goal of the recovery is to reproduce the tree structure, transforming the DS→BS correspondences into their calculated content.

Thanks to this feature, we do not have to handle independently multiple versions of the structures. The dependent form, within a MultiX document, ensures structure scalability and consistency. But the independent form can be useful for treatments; it is in particular better suited for query operations when there is no necessity to take into account the interactions between structures. Thus, MXQ can always provide updated and coherent structures of a MultiX document in the form of classic XML documents.

[13] http://liris.cnrs.fr/~peportie/multix/mxq.html

5.2 QUERYING MULTIX DOCUMENTS WITH XQUERY

When querying MultiX documents, multiple structures can be involved through their correspondences. The DS→BS correspondences can provide information about the content shared by various structures. This information can be obtained through the fragments of the basic structure. Moreover, the exploration of relations between documentary structures can be very helpful and always depends on the semantic of the document's domain.

Apart from the "rebuild" function introduced above, the MXQ library includes the following functions:

1. Functions handling DS→BS correspondences:

    a. `share-content($e as element()) as xs:Boolean`
    tests if the argument share some content with elements from other documentary structures than the one it belongs to. N.B. the special 'SP' fragment representing a single space is not considered in the process of finding shared content.

    b. `share-content-with($e as element(), $str_name as xs:string) as element()*`
    returns the elements from `$str_name` documentary structure pointing to a subset of the BS nodes composing the first argument.

    c. `share-fragments($e1 as element(), $e2 as element()) as xs:Boolean`
    tests if the arguments share some content.

    d. `get-shared-fragments($e1 as element(), $e2 as element()) as element(msd:frag)*`
    returns the fragments shared by the arguments.

    e. `include-fragments-of($e1 as element(), $e2 as element()) as xs:Boolean`
    tests if the content of the second argument is included in the first; the order of the fragments is not meaningful.

    f. `same-fragments($e1 as element(), $e2 as element()) as xs:Boolean`
    tests if the two arguments are equal in content.

    g. `include-content-of($e1 as element(), $e2 as element()) as xs:boolean`
    tests if the content of the second argument is included in the first; the order of the fragments being relevant.

2. Functions handling relations between documentary structures:

    a. `get-corr-target-from($e as element(), $label as xs:string) as element()*`
    returns the elements target of a DS→DS correspondence that originates in the first argument and is labeled by the second.

    b. `get-corr-target($label as xs:string, $d as document-node()) as element()*`
    returns the targets of a DS→DS correspondence that originates in any of the elements of the second argument and is labeled by the first.

c. `get-corr-source($label as xs :string) as element()*`
returns the elements source of a DS→DS correspondence labeled by the argument.

d. `exists-corr-between($e1 as element(),$e2 as element()) as xs:Boolean`
tests if there is a DS→DS correspondence from the first argument to the second.

e. `is-correspondence($e1 as element(), $e2 as element(), $label as xs:string) as xs:Boolean`
tests if there is a correspondence named by the last argument and linking the first to the second.

We now illustrate the use of these functions on our manuscript example.

Q1: Find all damaged words that only contain damaged characters.

```
xquery version "1.0";

import module namespace mxq = "multixlib" at "multixlibrary.xquery";

declare namespace msd = "http://www.msdm.org/2006/MULTIX";
declare namespace str2 = "http://exemple.org/words";
declare namespace str3 = "http://exemple.org/damaged";

<TEST xmlns:msd="http://www.msdm.org/2006/MULTIX">
{
let $doc := doc("manuscript.xml")

for $w in $doc//msd:DS[@name = "words"]//str2:w,
    $d in $doc//msd:DS[@name = "damaged"]//str3:dmg
where mxq:same-fragments($w, $d)
return
     mxq:rebuild($w)
}
</TEST>
```

Result:

```
<?xml version="1.0" encoding="UTF-8"?>
<TEST xmlns:msd="http://www.msdm.org/2006/MULTIX">
   <w>mid</w>
</TEST>
```

Q2: Find all the words that overlap two physical lines.

```
xquery version "1.0";

import module namespace mxq = "multixlib" at "multixlibrary.xquery";

declare namespace msd = "http://www.msdm.org/2006/MULTIX";
declare namespace str1 = "http://exemple.org/lines";
declare namespace str2 = "http://exemple.org/words";

<TEST xmlns:msd="http://www.msdm.org/2006/MULTIX">
{
```

```
let $doc := doc("manuscript.xml")

for $l in $doc//msd:DS[@name = "lines"]//str1:line,
    $w in $doc//msd:DS[@name = "words"]//str2:w
where mxq:share-fragments($l, $w) and not(mxq:include-content-of($l, $w))
return
      mxq:rebuild($w)

}
</TEST>
```

We use the fact that a word that splits on two lines only shares parts of its fragments with each line.

Result:

```
<?xml version="1.0" encoding="UTF-8"?>
<TEST xmlns:msd="http://www.msdm.org/2006/MULTIX">
   <w>ægþer</w>
   <w>spræce</w>
</TEST>
```

Q3: Find all words with restored characters. Indicate for each of them the restored characters and the region of the manuscript image where those characters can be seen.

```
xquery version "1.0";
import module namespace mxq = "multixlib" at "multixlibrary.xquery";
declare namespace msd = "http://www.msdm.org/2006/MULTIX";
declare namespace str1 = "http://exemple.org/lines";
declare namespace str2 = "http://exemple.org/words";
declare namespace str3 = "http://exemple.org/damaged";
declare namespace str4 = "http://exemple.org/text-regions";


<TEST xmlns:msd="http://www.msdm.org/2006/MULTIX">
{
let $doc := doc("manuscript.xml")

for $w in $doc//msd:DS[@name = "words"]//str2:w,
    $r in $doc//msd:DS[@name = "damaged"]//str3:res
where mxq:share-fragments($r, $w)
return
      <words-with-rest-char>
             {mxq:rebuild($w),
             <rest-char>{mxq:get-shared-fragments($r, $w)/text()}</rest-char>,
             for $l in $doc//msd:DS[@name = "lines"]//str1:line
             where mxq:share-fragments($r, $l)
             return mxq:get-corr-target-from($l, "localisation")
             }
      </words-with-rest-char>
}
</TEST>
```

Result:

```
<?xml version="1.0" encoding="UTF-8"?>
<TEST xmlns:msd="http://www.msdm.org/2006/MULTIX">
   <words-with-rest-char>
      <w>þu</w>
      <rest-char>þu</rest-char>
      <region xmlns="http://exemple.org/text-regions"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              num="reg.1"
              description="ligne 1">
        <zone xtop="43" ytop="50" xdown="460" ydown="94"/>
        <msd:anchor id="A.reg.1"/>
      </region>
   </words-with-rest-char>
   <words-with-rest-char>
      <w>me</w>
      <rest-char>m</rest-char>
      <region xmlns="http://exemple.org/text-regions"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              num="reg.1"
              description="ligne 1">
        <zone xtop="43" ytop="50" xdown="460" ydown="94"/>
        <msd:anchor id="A.reg.1"/>
      </region>
   </words-with-rest-char>
   <words-with-rest-char>
      <w>þines</w>
      <rest-char>s</rest-char>
      <region xmlns="http://exemple.org/text-regions"
              xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
              num="reg.3"
              description="ligne 3">
        <zone xtop="43" ytop="166" xdown="536" ydown="210"/>
        <msd:anchor id="A.reg.3"/>
      </region>
   </words-with-rest-char>
</TEST>
```

We have shown some applications of the MXQ library. Its main purpose was to demonstrate the exploitability of multi-structured documents. Work has been done in (Alink, Bhoedjang, Vries, & Boncz, 2006) to determine the best ways of extending XQuery in the context of multi-structured documents represented by stand-off markup; it could certainly be used to efficiently implement the MXQ functions.

6 CONCLUSIONS

In this paper, we have defined a formal model for multi-structured documents: MSDM. The logical model of MSDM is a graph. A basic structure organizes the content in disjoint elementary fragments. Documentary structures point to the basic structure by way of composition nodes. Correspondences can be modeled between a documentary structure and the basic

structure, as well as between distinct documentary structures. The model takes into account inter-dependencies of structures with shared content.

XML is a widely used language and the basis of several other formalisms. That's why we proposed MultiX, an XML instance of the MSDM model. It can therefore benefit from several existing XML tools and formalisms. Thus, for example, we can use the XSLT language to extract and format the documentary structures of a MultiX document. Moreover, we have developed a library of XQuery functions to easily explore multi-structured documents through elaborate queries. With these tools, we could now consider “Multi-Structured Information Retrieval”. This concept could be attached to the Web of Data in order to provide many points of view on the same raw content.

The creation of a multi-structured document from structures that share content is generally not an easy task for a human operator. The complete or partial automation of this task is sometimes necessary. We have already implemented the MXP parser (Multi-XML parser) (Chatti N. , 2006) in order to create an internal representation of a multi-structured document from XML structures encoded separately.

Finally, we have introduced the non-trivial problem of multi-structured documents construction. Although the XML enforcement of tree structures was considered for a long time the heart of the multi-structured document problem, we used it in the context of a new solution where the emergence of overlapping hierarchies triggers the creation of a new structure that has to be validated by a community of users. Thanks to the MSDM model, we have been able to address this new problem.

7 REFERENCES

Abascal, R., Beigbeder, M., Benel, A., Calabretto, S., Chabbat, B., Champin, P.-A., et al. (2003). Modéliser la structuration multiple des documents. *H2PTM* (pp. 253-258). Paris: Hermès.

Alink, W., Bhoedjang, R., Vries, A., & Boncz, P. (2006). Efficient XQuery Support for Stand-Off Annotation. *Proceedings of the 3rd International Workshop on XQuery Implementation, Experience and Perspectives, in cooperation with ACM SIGMOD.* Chicago, USA: ACM.

Bański, P. (2010). Why TEI stand-off annotation doesn't quite work. *Balisage: The Markup Conference 2010.* Montréal (Canada): http://www.balisage.net/Proceedings/vol5/html/Banski01/BalisageVol5-Banski01.html.

Bird, S., Buneman, P., & Tan, W.-c. (2000). Towards a Query Language for Annotation Graphs. *Proceedings of the Second International Conference on Language Resources and Evaluation*, (pp. 807–814). Athens (Greece).

Bird, S., Liberman, M., & Walker, M. (1999). Annotation Graphs as a Framework for Multidimensional Linguistic Data Analysis. *Towards Standards and Tools for Discourse Tagging: Proceedings of the Workshop* (pp. 1–10). Somerset, New Jersey: Association for Computational Linguistics.

Bruno, E., & Murisasco, E. (2006). Describing and querying hierarchical XML structures defined over the same textual data. *Proceedings of the 2006 ACM symposium on Document engineering* (pp. 147–154). New York, NY, USA: ACM.

Calabretto, S., Bruno, E., & Murisaco, E. (2007). *Documents and multiple hierarchies: Towards Multi Structured XML Documents.* Internal Research Report.

Chatti, N. (2006). Documents multi-structurés : de la modélisation à l'exploitation. *Thèse de Doctorat en Informatique*. Lyon: INSA de Lyon.

Chatti, N., Kaouk, S., Calabretto, S., & Pinon, J.-M. (2007). MultiX: an XML-based formalism to encode multi-structured documents. *Extreme Markup Languages.* Montréal (Canada).

Dekhtyar, A., & Iacob, I. (2005). A framework for management of concurrent XML markup. *Data & Knowledge Engineering vol. 52 num. 2*, 185-208.

Hilbert, M., Witt, A., & Schonefeld, O. (2005). Making CONCUR work. *Extreme Markup Languages.* Montréal, Canada.

Huitfeldt, C., & Sperberg-McQueen, M. (2003). *TexMECS An experimental markup meta-language for complex documents*. Consulté le 01 2011, sur http://decentius.aksis.uib.no/mlcd/2003/Papers/texmecs.html

ISO 8879. (1986). Standard Generalized Markup Language (SGML). International Organization for Standardization (ISO) - Information Processing – Text and Office Systems.

Jagadish, Lakshmanan, Scannapieco, Srivastava, & Wiwatwattana. (2004). Colorful XML: one hierarchy isn't enough. *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data* (pp. 251–262). New York, NY, USA: ACM.

Le Maître, J. (2006). Describing multistructured XML documents by means of delay nodes. *DocEng '06: Proceedings of the 2006 ACM symposium on Document engineering* (pp. 155–164). New York, NY, USA: ACM.

Nanard, J., & Nanard, M. (1995). Adding macroscopic semantics to anchors in knowledge-based hypertext. *International Journal of Human-Computer Studies*, 363-382.

Peroni, S., & Vitali, F. (2009). Annotations with EARMARK for arbitrary, overlapping and out-of order markup. *Proceedings of the 9th ACM symposium on Document engineering* (pp. 171-180). Munich, Germany: ACM.

Portier, P.-E., & Calabretto, S. (2009). Creation and maintenance of multi-structured documents. *Proceedings of the 9th ACM symposium on Document engineering* (pp. 181–184). Munich, Germany: ACM.

Portier, P.-E., & Calabretto, S. (2010). DINAH, a philological platform for the construction of multi-structured documents. *Proceedings of the 14th European conference on Research and advanced technology for digital libraries* (pp. 364–375). Glasgow, UK: Springer-Verlag.

Poullet, l., Pinon, J.-M., & Calabretto, S. (1997). Semantic Structuring Of Documents. *Basque International Workshop on Information Technology*, 118.

TEI Consortium. (2011). TEI P5: Guidelines for Electronic Text Encoding and Interchange. http://www.tei-c.org/Guidelines/P5/.

Tennison, J., & Piez, W. (2002). The Layered Markup and Annotation Language (LMNL). *Extreme Markup Languages.* Montréal (Canada).

Tummarello, G., Morbidoni, C., & Pierazzo, E. (2005). Toward Textual Encoding Based on RDF. *ELPUB, International Conference on Electronic Publishing* (pp. 57-63). Katholieke Universiteit Leuven in Leuven-Heverlee(Belgium): Peeters Publishing Leuven.

W3C. (2010). *XQuery 1.0: An XML Query Language (Second Edition)*. Retrieved from http://www.w3.org/TR/xquery/