Markup Overlap: Improving Fragmentation Method

Mohamadou Nassourou
Department of Computer Philology & Modern German Literature
University of Würzburg
Am Hubland
D - 97074 Würzburg
mohamadou.nassourou@uni-wuerzburg.de

Abstract: *Overlapping is a common word used to describe documents whose structural dimensions cannot be adequately represented using tree structure. For instance a quotation that starts in one verse and ends in another verse. The problem of overlapping hierarchies is a recurring one, which has been addressed by a variety of approaches. There are XML based solutions as well as Non-XML ones. The XML-based solutions are: multiple documents, empty elements, fragmentation, out-of-line markup, JITT and BUVH. And the Non-XML approaches comprise CONCUR/XCONCUR, MECS, LMNL ...etc.*
This paper presents shortly state-of-the-art in overlapping hierarchies, and introduces two variations on the TEI fragmentation markup that have several advantages.

Keywords: XML, Overlapping structures, Fragmentation.

Introduction

Overlapping is a common word used to describe documents whose structural dimensions cannot be adequately represented using tree structure. Non-hierarchical structures allow some children to share simultaneously several parents. For instance a verse that starts in one page and ends in the next one.
The problem of overlapping hierarchies is not a new one, and has been addressed by several approaches. There are Extensible Markup Language (XML) based solutions, as well as Non-XML ones. The XML-based solutions are multiple documents approach, empty elements, fragmentation, out-of-line markup, Just In Time Trees (JITT), and Bottom Up Vertical Hierarchies (BUVH), and the Non-XML approaches are CONCUR, XCONCUR, MECS, LMNL ...etc.

The Non-XML approaches introduce new syntaxes and semantics that need to be learned and understood. Not being XML constitutes a major disadvantage for these solutions.

The XML based solutions such as empty elements require customization of the existing standard XML tools in order to retrieve data enclosed by those empty elements. Multiple documents approach has unavoidable drawbacks such as data multiplication, and waste of storage space. JITT [11] cannot handle self-overlap, and validation across hierarchies is not possible. BUVH [10] has got the disadvantage of being computationally too intense. Stand-off markup is a tool dependent technique.

These disadvantages altogether require the modification of the standard tree structure of XML, which in turn could demand the development of new tools and even new algorithms.

Based on these drawbacks and the tendency to strictly maintain the tree structure of XML, I would suggest trying to eliminate disadvantages introduced by the fragmentation technique, because of its ability to cope with the XML structural ideology. It is worth noting that fragmentation provides a convenient method for handling overlapping structures, as far as logical order is concerned. Hence remodeling of the document logical structure could improve fragmentation technique. In the first method that I am proposing, remodeling of the document through empty elements is employed. The second proposal requires virtual extension of the TEI tagset. Both proposals eliminate the need for explicitly joining fragments in order to reconstitute the element they represent. In fact each fragment requires at least two attributes an id, and a pointer to previous/next fragment. These attributes are used to reconstruct the fragmented element. The proposed methods reduce the number of needed attributes, and enable the retrieval of the fragmented element without explicitly joining the fragments. As consequence, overall performance of XML-based solutions will be improved.

Non-XML approaches

These approaches introduce new syntaxes and semantics whereby elements do not need to form hierarchical structures as in XML. The best known probably are Multi-Element Code System (MECS) and Trivially Extended MECS (TexMECS) developed by the University of Bergen [7], and the Layered Markup and aNnotation Language (LMNL) [9]. In [7] a data structure known as General Ordered-Descendant Directed Acyclic Graph (GODDAG) for representing documents with overlapping structures was proposed. SGML [3], which is actually the forerunner of XML, also provided the CONCUR feature for representing non-hierarchical structures. XCONCUR has been proposed by [13, 14] as a way of improving and implementing the SGML CONCUR feature. Not being XML constitutes a major disadvantage for these solutions. XML is an easy, human readable language that has tools available for processing it. XML users would definitely prefer solutions within XML itself than new syntaxes.

Even though these techniques have solved the problem in their own ways, they do still have some disadvantages such as:

a) Inability to completely deal with self overlapping
b) Full validation is not possible.
c) XML schema needs to be modified.
d) Standard XPath/XQuery cannot extract information from the markup.
e) Finally, there is need to learn these techniques before using them.

XML-based Solutions

Overlapping structures cannot be easily represented using XML due to the fact that, XML has got a tree data structure. A child can have only one parent or be the parent itself.
Before delving into the Text Encoding Initiative (TEI) proposed solutions, I would like to present a short analogy of overlapping issue from our daily life.

How to solve a problem whereby a child belongs to more than one father?
A judge might decide to select one of the following options:
1) Let them have the child simultaneously by living together: not easy because child's father not known, and he might not be properly educated because of parents' conflict.
This sounds like milestones.
2) Let each one have the child for sometime: need to define time?
This is Multiple documents approach: one original and several reference documents.
3) Keep the child in different place and allow each one to come and see him for sometime: time has to be defined?
This case sounds like fragmentation and join.
4) Clone the child and give to each one of them: so who will get the original one?
This is Multiple documents approach: original one is copied several times.
5) No one will get the child but they can see him from far: so who will get the child?
This represents stand-off markup.

None of the above options will satisfy any of the fathers, as long as each one is longing to have the child alone.

Throughout this paper, I will be using the following two verses of chapter 36 (Ya-Sin) from the Quran (the Holy book of Islam), containing a quotation split between the two verses.
Verse 24:
"I would indeed, if I were to do so, be in manifest error.
Verse 25:
For me, I have faith in the Lord of you (all): listen, then, to me!"

Marking up the grammatical view of the verses we get:

```
<book><chapter>
<verse n="24"><q>"I would indeed, if I were to do so, be in manifest error.
</verse>
<verse n="25">For me, I have faith in the Lord of you (all):
 listen, then, to me!"</q>
</verse>
</chapter>
</book>
```

Overlapping has occurred. The q element is divided between the two verses.
To remediate this problem the TEI [1] proposes several solutions, which are not sufficient due to their disadvantages in one way or the other.
TEI guidelines for dealing with overlapping are:

1. Empty elements to delimit the overlapping elements.
2. Fragmentation to break down elements into smaller sections which do not overlap.
3. Stand-off markup to separate markup and document content.
4. Multiple documents approach to encode each hierarchy separately.

Case 1, 2, and 4 are known to be in-line markup solutions because markups are inserted into the document, while case 3 is known as the out-of-line markup in the sense that, it separates markups with the document content.

Multi-colored trees [12], JITT (Just In Time Trees) [11], and BUVH (Bottom Up Vertical Hierarchies) [10] are XML based solutions but have not been strongly supported by TEI.

In this paper, variations of the standard fragmentation method have been proposed. So let's state the problem that needs to be addressed.

Problem Specification

The problem can be generalized as follows: how to retrieve any given element without explicitly joining the fragments?

To demonstrate my proposals I am going to use the previous example.
Fragmentation recommends that one of the document hierarchies be selected as primary, and overlapping elements be modified by splitting them so that, they fit properly within the hierarchy with the possibility of virtually reconstituting them. Reconstitution of virtual elements is usually combined with the fragmentation method.

Marking up our example with fragmentation we get:

```
<book><chapter>
<verse n="24"><q id="q1" next="#q2">
"I would indeed, if I were to do so, be in manifest error.</q>
</verse>
<verse n="25"> <q id="q2" prev="#q1">For me, I have faith in the Lord of
you (all):  listen, then, to me!"</q>
</verse>
</chapter></book>
```

The attributes next and prev help to reconstitute the element by joining the fragments accordingly. TEI provides 'part' attribute as well for reconstructing the fragmented element.

The drawbacks of this method are:
a) Parser needs to know how to order the fragments
b) Number of fragments can grow up seriously
c) Number of fragments can be misleading about the actual instances of the element.

Selective AUgmented Fragmentation (SAUF)

The idea of SAUF comes from the fact that, a word that does not fit at the end of a line of text could be handled in one of the following ways:

a) split the word into two, the first part with an added hyphen remains on the current line and the second part is written on the following line.
b) write the word on the current line by using smaller font.
c) leave the space empty at the end of the current line, and write the word on the next line.

Case (a) represents the standard fragmentation because the word is split, and its parts could be joined using some attributes like id, next/prev.
SAUF is a way of implementing case (c) and (b).
Above I used a word as an example; however it could be a collection of words as well.

SAUF combines fragmentation, milestones and document remodeling techniques to eliminate the drawbacks of the fragmentation method.
It is called Selective AUgmented Fragmentation because one of the fragments is selected, and its text node augmented.
Fragments that are reduced to milestone elements could be seen as virtually non-empty milestones because of their metric attributes. Following paragraphs will clarify this situation.

Fig. 2 shows a graphical representation of SAUF.

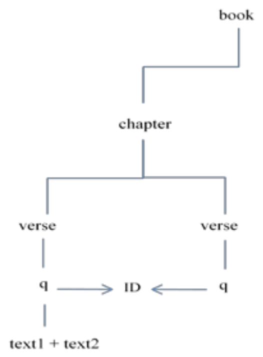

Fig. 1. SAUF graphical view

The algorithm to be used is explained with our previous example:

1) Let the first fragment get the whole element

```
<book><chapter>
<verse n="24"><q id="q1" next="#q2">
"I would indeed, if I were to do so, be in manifest error. For me, I have faith
in the Lord of you (all):  listen, then, to me!"</q>
</verse>
<verse n="25"> <q id="q2" prev="#q1">For me, I have faith in the Lord of
you (all):  listen, then, to me!"</q>
</verse>
</chapter></book>
```

This introduces a repetition of the second verse. Adding a copyOf/sameAs attribute in verse 25, we virtually eliminate the repetition.

```
<book><chapter>
<verse n="24"><q id="q1">
  "I would indeed, if I were to do so, be in manifest error. For me, I have faith
    in the Lord of you (all):  listen, then, to me!"</q>
</verse>
<verse n="25"> <q id="q2" copyOf="#q1"/>
</verse>
</chapter></book>
```

```
<book><chapter>
<verse n="24"><q id="q1">
"I would indeed, if I were to do so, be in manifest error. For me, I have faith
in the Lord of you (all):  listen, then, to me!"</q>
</verse>
<verse n="25"> <q id="q2" sameAs="#q1"> For me, I have faith
  in the Lord of you (all):  listen, then, to me!"</q>
</verse>
</chapter></book>
```

Remarks:
With this **semi-SAUF** method, it is possible to retrieve the whole quotation.
However the size of the document increases which is not desirable. Moreover the actual content of the augmented verse is no longer retrievable. And searching for verses containing only “faith” will produce erroneous result.

Therefore this increase in size must be solved with the help of additional attributes.
The solution is to make the second fragment a Trojan milestone, and add to both fragments two attributes: startPos and Length to specify the text of each verse. Both startPos and Length are normalized Unicode character offset.
The empty fragment is assigned an IDREF attribute pointing to the selected fragment where to find its text node.
We get the following:

```
<book><chapter>
<verse n="24"><q id="q1" startPos="0" Length="m">
"I would indeed, if I were to do so, be in manifest error. For me, I have faith
in the Lord of you (all):  listen, then, to me!"</q>
</verse>
<verse n="25"><q IDREF="q1" startPos="m+1" Length="n"/></verse>
</chapter></book>
```

The values "m" and "n" in Length="m" and Length="n" represent the actual length (normalized Unicode character offset) of the verse 24 and verse 25 respectively. These values could be manually calculated. However for the sake of accuracy, an XML editor supporting SAUF would be recommendable for automatically generating them. Moreover if any of the verses is altered, the editor must instantly detect it and adjust the values accordingly.
It might be interesting to notice that, SAUF uses character offset just as stand-off markup does.

An XML editor implementing this method is under construction.

With this markup it is possible to retrieve the whole quotation as well as individual verses. However it won't be possible to retrieve individual verses without knowing the startPos and Length attributes. Therefore it is recommended to consult the XML Schema to get full information about the fragments.
The q fragment in verse 25 is what I am calling a virtually non-empty milestone element.
For instance using XPath notation one could write:

To obtain the whole quotation, one needs to use the fragment which has been selected to hold the whole text. In this case it is the one in verse 24.
//verse[@n=24]/q

To retrieve any particular verse, one has to get the attributes startPos and Length of the verse, and then retrieve the text from the selected fragment, starting from startPos till the character whose offset is equal to startPos + Length.
With XPath notation:
//verse[@n=25]/q[@startPos] = m,
//verse[@n=25]/q[@Length] = n,
then from the text of the selected fragment, get the text starting from startPos till the (m+n)th character offset.
The substring function could be used to retrieve the text.

The same procedure is valid for verse 24.

Now the question is, how to find out which fragment holds the whole text of the overlapping element?
Well, by simply checking the startPos attribute. Normally the fragment whose startPos value is the lowest is assumed to be the selected one.

Is this method applicable to other overlapping structures besides quotes?
Yes, following is another example involving line/sentence overlap:

Let's use a citation from René Descartes**:**
"Je pense donc je suis."

Suppose that we get a document in which this citation starts on one line and ends on the next one. "Je pense donc" is on the first line, and "je suis" on the next line.

Marking up both the physical and grammatical views of the document yields the following:

```
<l n="1"><s>Je pense donc</l>
<l n="2">je suis</s></l>
```

The sentence overlaps with the lines.
Fragmenting it with SAUF, we get:

```
<l n="1"><s id="s1" startPos="0" Length="13">Je pense donc je suis</s>
</l>
<l n="2"><s IDREF="s1" startPos="14" Length="7"/></l>
```

To retrieve the whole sentence, we use the first line
//l[@n=1]/s

To obtain the exact text of the first line, we need to get the values of startPos and Length attributes.
Using XPath:
//l[@n=1]/s[@startPos] = 0
//l[@n=1]/s[@Length] = 13
The actual text size is: 0+13 = 13
So the first 13 characters of the selected fragment constitute the exact text of the line. The substring (text(),0,13) could be used to perform the task.
To obtain the text of the second line, we follow the same procedure.
//l[@n=2]/s[@startPos] = 14
//l[@n=2]/s[@Length] = 7
from the text of the selected fragment, get the text starting from the 14th character till the 21th character (14+7) using substring (text(),14,21)**.**

Another example using the same citation by assuming that, it is a quotation.
The mark up looks like this:

```
<l><s><q>je pense donc</l>
 <l>je suis</q></s></l>
```

Fragmenting it with SAUF, we get:

```
  <l n="1" ><s id="1" startPos="0" Length="13">
               <q  id="1" startPos="0" Length="13">je pense donc je suis</q>
             </s>
   </l>
   <l n="2" ><s IDREF="1" startPos="14" Length="7">
                  <q  IDREF ="1" startPos="14" Length="7"/>
             </s>
   </l>
```

To obtain the whole quotation
//l[@n="1"]/s/q

To obtain the exact text of the first line, we need to get the values of startPos and Length attributes.
Using XPath:
//l[@n=1]/s/q[@startPos] = 0
//l[@n=1]/s/q[@Length] = 13
The actual text size is: 0+13 = 13
So the first 13 characters of the selected fragment constitute the exact text of the line. The substring (text(),0,13) could be used to perform the task.

To obtain the text of the second line, we follow the same procedure.
//l[@n=2]/s/q[@startPos] = 14
//l[@n=2]/s/q[@Length] = 7
from the text of the selected fragment, get the text starting from the 14th character till the 21th character (14+7) using substring (text(),14,21)**.**

Advantages of the Method:

a) Existing XML technologies can process the document without any special customization.
b) No need to reconstitute the element from the fragments.
c) Statistically there is only one fragment that contains the text.
d) It is understandable and manually editable.

Disadvantages:
Of course it has got disadvantages as well, like choosing arbitrary one hierarchy as the primary one. However this selection is based on the user’s intention. The user decides which element is a parent, a child, and so on.

Remarks:
One could argue that this method is not better than the standard fragmentation, because the physical structure of the document has been altered by expanding the content of verse 24 and deleting the content of verse 25. In fact <lb>, <pb>,<cb>,<handShift> proposed by the TEI are also altering the document physical structure. However as far as logical structure is concerned, it has indeed more advantages over the usual fragmentation method for the reasons listed above. Therefore to use fragmentation to its full strength, remodeling of the document logical structure must be employed as well.

Extended Fragmentation (ExFrag)

ExFrag disagrees with the tag naming of fragments, and proposes an extension of the TEI tagset by allowing virtual addition of suffixes to existing TEI tags. It is actually a policy and rule-based method.
As mentioned earlier the q fragments are given the very same name of the element they represent, even though each one contains only a portion of the text node. It would be more intuitive to denote them with q1 and q2 which could indicate at the level of syntax that, they are fragments of a q element.

Fig. 2 shows a graphical representation of ExFrag.

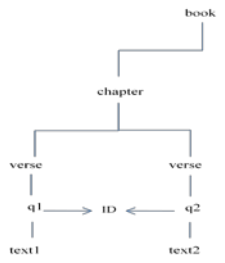

Fig. 2. ExFrag graphical view

A generalization of the method looks like this:

If any element overlaps with N other elements, fragment it N times, and add a suffix to each element numbering from 1 to N.

TEI Schema extension requires an addition of a global <any> element which should be a combination of a TEI tag with a digit. XML processors could split the element and identify which TEI tag it corresponds to.

It must also be clearly mentioned in the documentation that a TEI tag should only be adjoined with a digit if fragmentation occurs. For instance a <q1> element without <q2> should generate an exception error.

With XML schema the <any> element looks like this:
<xs:any minOccurs="0"/>

Using our previous example:
```
<book><chapter>
<verse n="24"><q1 id="q_verse_24_25">
"I would indeed, if I were to do so, be in manifest error. “</q1>
</verse>
<verse n="25"> <q2 IDREF="q_verse_24_25">For me, I have faith in the
Lord of you (all):  listen, then, to me!"</q2>
</verse>
</chapter></book>
```

The first fragment is assigned an id which is referenced from the remaining fragments using IDREF attribute. The id value q_verse_24_25 says that it is a q element divided between verse 24 and 25.
XML processors could implement this feature, and automatically reconstitute any element through its id attribute. Of course existing XML processors need to be extended to have the capability of automatic reconstitution.
However even with the currently available standards (XSLT, DOM …etc), ExFrag would be easier to process than the TEI fragmentation method, because it has got less number of attributes. In this method a parser needs to check the presence of a digit at the end of an element, and the id. While with the standard fragmentation a parser should either check the id, the prev and next attributes, or the id, initial, middle, and last attributes. Clearly it takes longer time and might even be more complicated to process.
In addition, with this method one is assured that the instances of an overlapping element cannot be erroneous, because the distinction between fragments and elements is obvious.

Using XPath:

//verse[@n=24]/q or //verse[@n=25]/q will produce the full quotation.

//verse[@n=24]/q1 or //verse[@n=25]/q2 will produce verse 24 and verse 25 respectively.

One more example with the citation of Descartes:

```
<p>
  <line n="1" ><s1 id="s_line_1_2">
                 <q1  id="q_line_1_2">je pense donc </q1>
               </s1>
   </line>
   <line n="2" ><s2 IDREF="s_line_1_2">
                    <q2  IDREF ="q_line_1_2"> je suis</q2>
               </s2>
   </line>
</p>
```

The id value s_line_1_2 says that it is an s element divided between line 1 and 2.
The id value q_line_1_2 says that it is a q element divided between line 1 and 2.

Retrieving with XPath:

//line[@n=1]/s/q or //line[@n=2]/s/q will produce the full quotation.

//line[@n=1]/s1/q1 or //line[@n=2]/s2/q2 will produce the corresponding fragment (or line text) respectively.

This method could be manually implemented. However an XML editor supporting ExFrag would be recommendable for automatically fragmenting and generating the ids accordingly. This would improve coding time and the markup efficiency.

An XML editor implementing this method is under construction.

Advantages:

The main advantage of this method is the fact that users or programmers do not need to bother about reconstructing fragmented elements. XML processors will automatically figure out and combine the fragments.
Moreover fragments are easily visible from the syntax itself.
The method has also achieved the aim of reducing the number of attributes, hence simplifying the process of segmentation and reconstitution. Finally, instances of an overlapping element can no longer be wrong.

Disadvantages:

New virtual tags have to be added to the TEI tagset.

Overlapping Markup Desiderata

Overlapping markup desiderata have been proposed by [5]. The paper argues that following points have to be fulfilled for any solution to be a perfect one.

- Adequacy
- Human readability
- Maintainability
- Available implementations
- XML compatibility
- Ease of validation
- Validation across hierarchies
- Ease of formatting
- Ease of extracting multiple views
- Ease of extracting hierarchical subsets
- Continuity of text content

The proposed methods comply with these desiderata, and are therefore good candidates for implementation in an XML editor.

Discussion and Conclusion

Each existing method of solving overlapping problem has got its pros and cons, therefore one should select the one that can best deal with the kind of overlap that occurs.

Fragmentation seems to be the most appropriate solution in the context of XML, because it is simple, human readable, and support all the existing XML technologies.

The proposed Selective Augmented Fragmentation (SAUF) and the Extended Fragmentation (ExFrag) methods have several advantages over the standard fragmentation as mentioned earlier. These methods eliminate mainly the need to recompose the fragmented elements.

Of course everything has got its limitations. While SAUF requires two new attributes (startPos and Length) in order to compute the exact text node of each fragment, ExFrag demands an extension of TEI tagset. However these limitations do not require some modifications of the XML tree structure ideology.

Referring to Non-XML solutions it would be a big win, if a simple and manually editable methods for converting their structures say MECS to standard XML could be developed.

References

[1] http://www.tei-c.org/release/doc/tei-p5-doc/en/html/NH.html
[2] W3C, Extensible Markup Language (XML). 2000.
[3] W3C, Standard Generalized Markup Language SGML). 1999.
[4] W3C, XML Path Language (XPath) Version 1.0. 1999.
[5] Steven DeRose. Markup Overlap: A Review and a Horse in Extreme Markup Languages 2004 (Montréal, Québec)
[6] Durusau, P. and M. O'Donnell. Implementing Concurrent Markup in XML. in Extreme Markup Languages 2001. 2001. Montreal.
[7] Sperberg-McQueen, C. and C. Huifeldt. GODDAG: A Data Structure for Overlapping Hierarchies. in ACH-ALLC'99. 1999. Charlottesville, Virginia.
[8] http://www.jenitennison.com/blog/node/97
[9] http://xml.coverpages.org/LMNL-Abstract.html
[10] Patrick Durusau, Matthew Brook O'Donnell. Concurrent Markup for XML Documents (Europe 2002)
[11] P. Durusau, M.Brook O'Donnell "Just-In-Time-Trees (JITTs):Next Step in the Evolution of Markup?" Proceedings of 2002 Extreme Markup Languages Conference, Montreal, Canada 2002
[12] Jagadish, H. V., Laks V. S. Lakshmanan, Monica Scannapieco, Divesh Srivastava, and Nuwee Wiwatwattana. 2004. “Colorful XML: One hierarchy isn't enough.” *Proceedings of the 2004 ACM SIGMOD International conference on management of data,* Paris
[13] Witt, Andreas. “Multiple hierarchies: new aspects of an old solution.” In *Proceedings of Extreme Markup Languages 2004*
[14] Mirco Hilbert, Oliver Schonefeld, Andreas Witt (2005). Making CONCUR work. In: Proceedings of the Extreme Markup 2005, Montréal, Canada
[15] Schmidt, Desmond. “Merging Multi-Version Texts: a Generic Solution to the Overlap Problem.” Presented at Balisage: The Markup Conference 2009, Montréal, Canada, August 11 - 14, 2009. In *Proceedings of Balisage: The Markup Conference 2009*. Balisage Series on Markup Technologies, vol. 3 (2009). doi:10.4242/BalisageVol3.Schmidt01.