Extreme Markup Languages 2004®

Montréal, Québec
August 2-6, 2004

Markup Overlap: A Review and a Horse

Steven DeRose
Bible Technologies Group

Abstract

"Overlap" is the common term for cases where some markup structures do not nest neatly into others, such as when a quotation starts in the middle of one paragraph and ends in the middle of the next. OSIS [Duru03], a standard XML schema for Biblical and related materials, has to deal with extreme amounts of overlap. The simplest case involves book/chapter/verse and book/story/paragraph hierarchies that pervasively diverge; but many types of overlap are more complicated than this.

The basic options for dealing with overlap in the context of SGML [ISO 8879] or XML [Bray98] are described in the TEI Guidelines [TEI]. I summarize these with their strengths and weaknesses. Previous proposals for expressing overlap, or at least kinds of overlap, don't work well enough for the severe and frequent cases found in OSIS. Thus, I present a variation on TEI milestone markup that has several advantages. This is now the normative way of encoding non-hierarchical structures in OSIS documents.

Markup Overlap: A Review and a Horse

Table of Contents

Markup Overlap: A Review and a Horse

Steven DeRose

§ Introduction

"Overlap" describes cases where some markup structures do not nest neatly into others, such as when a quotation starts in the middle of one paragraph and ends in the middle of the next. OSIS [Duru03], a standard XML schema for Biblical and related materials, has to deal with extreme amounts of overlap. The simplest is book/chapter/verse and book/story/paragraph hierarchies that pervasively diverge; but many types of overlap are more complicated than this.

The basic options for dealing with overlap in the context of SGML [ISO 8879] or XML [Bray98] are described in the TEI Guidelines [TEI]. I summarize these with their strengths and weaknesses. Previous proposals for expressing overlap, or at least kinds of overlap, don't work well enough for the severe and frequent cases found in OSIS. Thus, I present a variation on TEI milestone markup that has several advantages, though it is not a panacea. This is now the normative way of encoding non-hierarchical structures in OSIS documents.

§ The problem types

The simplest type of non-hierarchical structure is the union of multiple structures, each of which is hierarchical. In this case, if you discard all the markup for all but one of the structures, the one structure remaining is a non-problematic hierarchy. For example, the Biblical book/chapter/verse and book/story/paragraph/sentence hierarchies are essentially independent below the level of book (which must be identical across the hierarchies).

A particularly nasty case of such overlap arises when one kind of component crosses many others, as sometimes happens with quotes. The example below shows a typical case from the Biblical book of Jeremiah, and one way to encode it by segmenting the non-hierarchical quotes into parts that are hierarchical, then co-indexing the parts of each logical quote by assigning them the same value for the "splitID" attribute.

First, consider the passage without markup. The parenthesized numbers indicate where each numbered verse begins; indentation indicates nested quotations.

```
(1) Moreover the word
of the LORD came to me, saying,

   (2) Go and cry in the
   hearing of Jerusalem, saying,

      Thus says the LORD:
         I remember you,
         The kindness of your youth, The love of your
         betrothal, When you went after Me in the
         wilderness, In a land not sown.

         (3) Israel [was] holiness
         to the LORD, The firstfruits of His increase.
         All that devour him will offend;
         Disaster will come upon them,

       says the LORD.

(4)....
```

The difficulty is that verse 3 begins in the middle of a third-level nested quote. In a hierarchy, ending verse 2 forces ending the three quotes as well. Then as verse 3 begins all those quotes must be re-opened. Further, it is important to encode somehow that the three quotes opened at the start of verse 3 are not just any three quotes, but exactly the same ones that were closed to end verse 2. Put another way, these quotes should not logically have closed at the verse boundary at all -- if they close it is because of syntax limitations. Below is the marked-up equivalent, where distinct physical parts of each single logical quote are co-labeled using values of the splitID attribute:

```
<div>
  <p>
    <verse osisID="Jer.2.1">Moreover the word
```

```
      of the LORD came to me, saying,</verse>
    <verse osisID="Jer.2.2">
      <q splitID="Q-Jer.2.2-A">Go and cry in the
              hearing of Jerusalem, saying,
        <q splitID="Q-Jer.2.2-B">Thus says the LORD:
          <q splitID="Q-Jer.2.2-C">I remember you,
            The kindness of your youth, The love of your
            betrothal, When you went after Me in the
                      wilderness, In a land not sown. </q>
        </q>
      </q>
    </verse>
    <verse osisID="Jer.2.3">
      <q splitID="Q-Jer.2.2-A">
        <q splitID="Q-Jer.2.2-B">
          <q splitID="Q-Jer.2.2-C">Israel [was] holiness
          to the LORD, The firstfruits of His increase.  All
          that devour him will offend; Disaster
          will come upon them,</q>
        <!--True Close Q-Jer.2.2-C-->
        says the LORD.</q>
        <!-- True Close Q-Jer.2.2-B -->
      </q>
      <!--NO True Close Q-Jer.2.2-A -->
    </verse>
  </p>
...
```

This sort of case is commonly advanced to show that the "Ordered Hierarchy of Content Objects" model does not suffice [see Dura96, Rene95]. It does show that, but it remains simple enough that theories and systems of multiple simple hierarchies remain viable.

Somewhat more complicated are cases where components of what is arguably the *same* hierarchy can remain open beyond the end of the component in which they began, but instances of the same component type can never overlap each other. This may not seem to be a natural category because it depends on the details of particular schema; but it is a common class: even quotations operate this way. Quotes generally do not partially overlap other quotes, although they may overlap many other structures and may be arbitrarily nested. This model is very close to that of MECS [Sper98] and permits some markup syntax and processing simplifications. For example, there is no need to provide or track co-indexing for start and end-markers, because the element type is enough to permit correct match-up.

Yet worse are cases where instances of the same component type can overlap each other. The classic example of this is hypertext links or annotations, which overlap arbitrarily. These can be represented neatly by very powerful methods such as milestones, joins, and standoff markup (see below), but lead to an unreadable mess with most other methods (and, some would surely argue, even with these methods).

Finally, the worst case is where not only do components overlap, but some components are discontiguous. Such cases have not generally been considered part of the overlap problem. The most powerful tools for overlap (namely joins and standoff markup) also cover these cases, but impose considerable costs.

In OSIS documents, such cases arise mainly from translations into languages whose structures are particularly different from the Greek or Hebrew source languages. Semantic units such as paragraphs may span several verses. The rules of some languages prefer or sometimes require that information be presented in certain orders (particularly in stories and historical presentations). Thus, best practice in translation can lead to verses being split, with a part moved away from the rest; or even to verses being combined indissolubly. These cases pose difficult problems if the original structure is to be marked up in the translation. Such cross-edition markup can be valuable, for example for correlating across translations. However (fortunately, at present) this is not common practice: such movements are usually done without comment, or with a prose footnote. They are not commonly marked up at all. It is possible, however, that part of the reason they are not marked up is the syntactic difficulty of doing so. Discontigous components will not be considered further here, but they will likely become an important issue once the simpler cases of markup overlap have been satisfactorily resolved.

§ Evaluation criteria

Several prior authors have set out criteria for identifying a "good" representation for overlap [Duru02b; Rene95]. The first is adequacy. However, as with many other markup problems (and computing problems in general), one can force-fit adequacy into even very sparse syntax. The classic example is the one-element DTD that can represent (though not validate) anything any other DTD can:

```
<!ELEMENT thing    (#PCDATA | thing)*>
<!ATTLIST thing    class CDATA #REQUIRED>
```

Comparable solutions are available for loading XML into databases, for computer processor instruction sets, and so on. Adequacy is necessary but not sufficient. Other important criteria include:

1. Human readability -- this rules out binary formats, where the meaning of a character is not fully obtainable from its glyph but requires its binary representation as well.
2. Maintainability -- this argues against representations that duplicate data or that use file offsets or binary hash.
3. Available implementations
4. XML compatibility
5. Ease of validation -- It is helpful if a syntax can make use of what validation XML can do, even though solutions to overlap are generally difficult to validate completely using generic XML software.
6. Validation across hierarchies -- this is a hard theoretical problem, and has not been addressed by most overlap solutions. The semantics of such validation are not obvious -- one cannot simply declare that component type A may contain components of type B; what does one say about B’s that partly overlap (at start or at end), or are co-located with either end of A? Durusau [Duru02b] and HyTime [ISO 10744] both enumerate 13 possible relations between two contiguous ranges -- though see below for a simplification of this model.
7. Ease of formatting -- both CSS and XSL assume documents are tree structures (or Ordered Hierarchies of Content Objects [DeRo90]): thus properties (such as font) are neatly inherited down the tree. When a portion of content instead has ancestors that overlap, the order in which inheritance occurs (if at all) is not obvious. Perhaps the simplest solution is the familiar "OLIST" discipline: components are added to an "open" list in the order they are encountered, and deleted from that list when they end (without causing any later components to be deleted); the OLIST can be used for inheritance with reasonable effects.
8. Ease of extracting multiple views
9. Ease of extracting hierarchical subsets
10. Continuity of text content

§ Some available solution types

SGML CONCUR

```
  <(DTD1)p>And the Lord said,
  <(DTD2)q>Read my lips: Do not murder.</(DTD1)p>
    <(DTD1)p>Be nice to each other instead.</(DTD2)q>
  And the people said "Amen."</(DTD1)p>
```

An SGML parser that supports CONCUR (it is rumored there is only one), will treat the markup from each DTD separately. It could either ignore all but one DTD on each pass, or parse them simultaneously but keep separate track of what elements are open in each.

The main advantages of CONCUR are that it is a part of SGML, and that it is quite legible and maintainable.

One key disadvantage is that CONCUR provides no way to constrain relationships across DTDs (nor schemas, for SGML only supports DTDs). For example, expressing and validating that book elements must synchronize in all DTDs, or that quote elements may cross paragraph elements but not chapter elements in the other DTD.

A second disadvantage is that CONCUR does not suffice for self-overlap (cases where two element of the same element type overlap). To handle self-overlap with CONCUR would require an unpredictable number of DTDs, depending on just how heavily things overlap. Also, those DTDs are completely unintuitive: for example, not all hyperlinks would be in the same schema: any that overlapped a hyperlink from DTD 1 would have to move to DTD 2; any that still overlapped would move to DTD 3, and so on.

Finally, CONCUR is not XML-compatible, and little if any current SGML software supports it; so it is not a widely usable solution.

An anonymous reviewer pointed out that although it is not a practical general technique, in some cases SGML OMITTAG can also be pressed into service. Clever adjustments can be done such that in one DTD a non-empty element is opened and later automatically closed, while in another the element is simply EMPTY.

Segmentation

Segmentation, as shown in the long example above, involves breaking up any overlapping elements into smaller pieces that do not overlap. The TEI Guidelines [TEI] describe this method in some detail.

This solution has several disadvantages. One is that one component must be made primary and the other secondary even though there is generally no principled basis for choosing. For example, OSIS users differ (sometimes strongly) on whether the reference hierarchy or the discourse hierarchy should be primary. Also, sometimes a component must be broken up into many pieces, such that it becomes hard to see the whole. Early versions of OSIS used segmentation, and users found such cases inscrutable. Finally, unless the parts of a single component are co-indexed (such as by the TEI "next" and "prev" attribute that link the component's segments into a list), there is no way to distinguish a segmented component from two components that happen to be adjacent, or that overlap each other in complex ways.

MECS [Sper98]

MECS defines three types of "codes": one equivalent to XML empty elements; one equivalent to elements with #PCDATA content; and one equivalent to elements with element content. MECS does not allow an element to overlap others of the same type. Because of this limitation, it can be implemented simply by changing the open element stack discipline of SGML or XML to a open element list model. When an element closes, it is removed from the open list, regardless of whether it is at the end of the list or not; all other elements on the list stay there.

The main problem with the MECS model is the inability to handle self-overlap. It also is a non-XML syntax, and so cannot take advantage of existing XML software and infrastructure. An additional problem is that it rules out most tag-omission possibilities; however, since XML rules out all such, many would not consider this a major issue.

JITTs

JITTs [Duru02, Duru02b] is similar to MECS in that is does not precisely use XML syntax (although it is very close). On the other hand, JITTs is similar to CONCUR in that it provides a way for the parser to only "count" some tags, not all tags. The fundamental advancement JITTs brings is that markup is recognized or ignored at parse-time rather than at encoding-time as with CONCUR. During parsing, the parser compares each tag found against a filter. Only those which the filter selects are returned to the application as real start or end tags; others act as if they had not been there at all (or perhaps are passed back as some non-tag kind of event). For example, a simple filter might accept all elements except <q>; or perhaps only <q>. CONCUR falls out as a special case, where the filter accepts only tags with a certain prefix (in XML, the prefix would presumably be a namespace prefix rather than a parenthesized DTD name).

JITTs' main insight is that a file need not be well-formed until the moment it is being processed. With a very small change to an XML parser (namely, inserting a filter that operates whenever a tag is recognized), documents that are not well-formed XML because of overlap can still be parsed and even validated subject to a given filter. Thus, one pass might parse and analyze just quotes, while another just recognizes linguistic structures, and another just recognizes chapter and verse references.

Like other solutions, JITTs does not provide a particular way to correlate and validate across structures. However, some kinds of relationship can be validated by constructing clever filters, and other relationships can be easily checked if a uniform way to refer to content locations is used (even one as primitive as raw file or entity offsets).

The main drawback of JITTs is that arbitrary self-overlap cannot be made well-formed by eliminating a simple subset of markup. Again taking the classic example of hypertext links, there is no principled way to predict which links must be ignored in order to eliminate all non-well-formedness. Workable subsets can be calculated for each actual document, but not in general. Even if there were a simple solution, the result would still be problematic because one may wish to be able to process all hypertext links at the same time.

A subtler problem is that if the filter recognizes some but not all elements of a single element type, it is not clear how to decide which end-tags should be ignored. Presumably it should be the ones that correspond to the start-tags that were ignored; but how is that, in turn, decided? This problem may also limit JITTs' effectiveness for self-overlap.

Joins

The TEI join element can point to any number of other elements, and explicitly assert that they are multiple physical parts of one logical whole. This construct is a variation on segmentation. It does not provide any way around the syntactic overlap problem, and authors must segment any overlapping elements as before. However, with join the specific relationships of the segments can be preserved, eliminating the ambiguity problems cited earlier.

In the example below, taken from [TEIP4], three portions of content, specified by their IDs, are combined into a new logical whole:

```
<sp who="hughie"><p>How does it go?
  <q><l id="x1">da-da-da</l>
     <l id="l2">gets a new frog</l>
     <l>...</l>
  </q></p></sp>
<sp who="louie"><p><q><l id="l1">When the old pond</l> ...</q></p></sp>
<sp who="dewey"><p><q>... <l id="l3">It’s a new pond.</l></q></p>
<!-- ... -->
<join targets="l1 l2 l3" result="lg" desc="haiku" scope="root"/>
</sp></para>
```

Note: With the XPointer xpointer() scheme (a W3C Working Draft, [XPtr01]), certain HyTime Location modules constructs [HyTime], or similar methods that can point to general ranges rather than merely whole elements, segmentation can be avoided by pointing directly to the range (or ranges) needed without marking each segment as an element first.

The main advantage of join is power: virtually any conveivable structure can be handled via join. Even discontiguous and/or re-ordered components can be expressed unambiguously. With some extra attributes, a join could even express partial-orderings [see Liu77] among physical components.

The main disadvantage of join is that the join object is not (in general) contiguous with the segments it is joining. This poses maintenance problems: it is too easy to modify the segments without updating the join or vice versa, and too hard to catch such errors because they almost always produce a valid (though incorrect) result. This is a standard problem of extremely powerful formalisms: they permit so much that they have trouble ruling out errors -- just as if every combination of English words was grammatical it would be hard to detect ungrammatical sentences.

Another disadvantage is that since it may still require segmentation, a single overlapping component may require a large number of tags and elements to represent it. This reduces readability and maintainability. In addition, the exact number of segments (and thus the number of entries listed in the join element) can change due to changes not involving the segmented element itself, such as inserting an element that is a child of an element the join overlaps.

Standoff markup

Standoff markup (sometimes also called "out of line" markup) includes methods that locate at least some markup quite separate from the content it applies to. Thus the TEI join structure just described is one form of standoff markup. However, standoff markup can be done in many other ways.

The W3C XLink Recommendation [XLink01] can be used to express links between any number of endpoints (or "anchors"), and to assert roles and relationships between them; those links can be expressed in a completely different location from any of their anchors. Standoff markup is the reason that XLink provides for links with only a single anchor: "single-ended" links. In such cases, the role of that anchor is generally used to assert something about it, just as XML elements are generally used to assert something about their content. For example, this XLink attaches the role "quote" to the elements with IDs l1, l2, and l3, without providing traversal to any other link-ends:

```
    <link xlink:type='extended'>
       <anchor xlink:type='locator' xlink:role='quote' href="#l1">
       <anchor xlink:type='locator' xlink:role='quote' href="#l2">
       <anchor xlink:type='locator' xlink:role='quote' href="#l3">
    </link>
```

HyTime [DeRo93], RDF [RDF04], Topic Maps [TM01], and "Reified" LMNL [Tenn02] can be used in similar ways.

Non-XML representations can also be used for standoff markup. One trivial method uses a file or a database relation with three fields in each record: start-offset, end-offset, and element type name; each record asserts the meaning of the element type name for the range specified:

```
    1    10   Q
  200   400   Q
 2047  4095   Q
   10    25   EMPH
 ...
```

This trivial method cannot express discontiguous components, but with a little modification it can. For example:

- a fourth field could group records identifying each contiguous part of a component by assigning them matching identifiers.
- next and/or prev fields could be added to link discontiguous parts into a list (this allows ordering the parts if desired).
- each record could be permitted multiple start/end offset pairs

Like joins, the main advantage of standoff markup is the power to express nearly any construct imaginable.

Again like joins, the main disadvantage of standoff markup is the difficulty of maintenance. File offsets break with even tiny edits of the content file. If the content file has some (non-standoff) markup, breakage can be reduced by using XPointer child sequences instead (an optional ID, followed by a series of child-numbers to walk down the tree to the right place [XPtr01]). Special markers could even be inserted to provide IDs at all needed points in content.

One often hears that such IDs are somehow "safe" pointers into documents. However, this is not true; they are at most "safer" than many other methods. Even with IDs, there is no guarantee that references will not break -- one could even write a program to change all the IDs on a Website daily. This would be unusual behavior in general, but might be used, for example, as a way to prevent outsiders from creating links to internal parts of a Website.

TEI-style Milestones

TEI provides, among other things, the "milestone" method for marking up non-hierarchical structures: In short, replace any troublesome element with a pair of empty elements, and put co-indexing values on them. For example:

```
    <p>Good morning.
    <milestone type='start' gi='q' id='q37'/>Be of good cheer</p>
    for today, at least.<milestone type='end' gi='q' id='q37'/>
```

The "gi" attribute provides the element type name for which the milestones elements are standing in. The id attributes cannot be of SGML type ID, because ID values must be unique; they are typically an ID/ IDREF pair.

The main advantages of milestones are that they keep the markup with the content (enhancing maintainability), while having enough power to express any contiguous components. Discontiguous components can be supported by adding methods such as next/prev or joins. Milestone markup conforms to XML and can be *parsed* with existing XML tools.

The main disadvantage of milestones is, however, that their XML-processability is limited. XML knows about the endpoints, but does not know that they are related or that they delimit a component that should be treated basically like elements. For example, both CSS and XSL use inheritance to apply formatting properties to the entire content of a given XML element, but they cannot readily do so for milestone-delimited ranges because they only know about true XML elements.

It is worth noting that quotations, which provide some of the most complex cases of overlap, are less problematic in terms of formatting. It is often enough merely to issue quotation marks at the start and end, and this does not require inheritance.

In my opinion, milestones constitute the most adequate method of dealing with overlap that is still human-maintainable (admitting that both adequacy and readability are subjective goals). The rest of this paper attempts to refine the use of milestones for representing overlap.

§ Differing type of milestones

There are several choices in the details of how milestones are used. The two most basic forms differ in whether they use a single element type with some attribute to distinguish start from end, or use distinct element types for start and end:

```
<milestone type='start' gi='q' id='foo'/>...<milestone type='end' gi='q' coid='foo'/>

<start gi='q' id='foo'/>...<end gi='q' coid='foo'/>
```

Both these have been proposed, including a variation of the second in an earlier version of OSIS. They both have the problem that there is no way to associate the right attributes with the generic milestone tags. For example in OSIS, all element types have a 'type' attribute, and for most it has an enumerated 'starter set' of values. Most existing schema mechanisms cannot validate that generic milestoned quotes get the same attributes as quote elements while milestoned verses get the same attributes as verse elements. However, as an anonymous reviewer pointed out, this does seem possible with RelaxNG [Clar01].

To solve this problem there must be distinct milestones for every milestonable element type. This works. Usually, such systems derive milestone type names systematically from the corresponding non-milestone elements. This is easy to learn, but triples the number of element types. Maintaining multiple declarations and their documentation (especially, keeping them in sync over time) can be error-prone, although sharing their declarations through parameter entities or schema types greatly reduces this problem.

```
q
q-start
q-end
```

A slight variation is to provide *only* milestones and no "regular" XML elements, but users dislike using milestones when they are not needed. Present software often can't give much help in using them either -- for example, CSS and XSL have no natural way to change the formatting of just the range between two such milestones, making editing tedious and error-prone.

§ Catching Hell'n tagging

These problems eventually led to the OSIS variation described here. Troy Griffiths, a principal technical member of the OSIS WG, floated the following argot at an OSIS meeting. Though Spartan, it sailed through the working group with hardly a ripple.

The solution used in OSIS is this: use neither a generic milestone tag nor specific extra ones, but instead use the very same tag as milestone and non-milestone elements: in empty and non-empty XML forms. A sighran through the group. In short,

- Use <q who='paris'>...</q> when you can, otherwise
- use <q who='paris' sID='foo'/>...<q eID='foo'/>

This approach, which after Troy is called "Trojan milestones", comes nearly for free with typical schema languages. No extra elements are needed. One need only add sID and eID attributes, and ensure that empty content is permitted. Ergo not any elements fail to be milestonable at will or at need. The proper attributes come automatically because the same declaration is in effect.

The advantages of Trojan milestones include those of milestones in general, with improved readability. They are also easy to teach (the milestoned form is, to the user, just a slight syntax variation on the normal form). A hierarchical subset of the markup can also be designed in, simply by choosing some elements that may not be "milestoned" (that is, which do not permit empty content, and/or do not permit the sID/eID attributes). For example, OSIS defines that the verse hierarchy is always secondary to the linguistic/rhetorical hierarchy, and so many tags in the latter are not milestonable. If counterexamples arise, the schema change to add them is entirely backward-compatible.

The advantage that (unlike generic milestones) Trojan milestones look like element tags (that is, they have the same GI) should not be underestimated; while unlike extra milestone elements with derived GIs, they do not expand the list of element types.

The main disadvantages are that like other milestone methods they are not fully processable by generic XML software; but in fact this is true of any form of overlap. The nearest thing to an exception is segmentation, where at least each segment of a component can be processed by typical XML applications; but even there the processors do not know (unless via complicated context-checking) that those segments are parts of a whole. For example, avoiding superfluous quotation marks or line breaks, performing containment searches correctly, and similar basic requirements are hard to fulfill.

As noted earlier, there are several Trojan milestone requirements that current schema languages will not enforce. These must be enforced at some higher level:

1. the element must be empty exactly when its sID or eID attribute is set (SGML could accomplish this with its much-maligned CONREF attributes).
2. when eID is present, no other attributes are permitted.
3. each sID/eID value should occur only twice (once on sID and once on eID)
4. empty elements with matching sID and eID values should match up in proper pairs and in order.

Note that because of the second rule above, no attributes may be *required* for milestoneable elements. Schema languages that can make attributes optional or required depending on the presence of other attributes (in this case eID) do not suffer this problem.

This model, originally named HORSE (Hierarchy-Obfuscating Really Spiffy Encoding) may seem wooden at first, but soon it comes to look quite natural. With its spartan syntax it should be attractive; but its name seems less so, and clearly a Muse meant us to rename this model

```
                              CLIX
```

because of its heavy use of point events scattered throughout the text or data stream: click, *clicks*, **clix**.

§ CLIX and LMNL

The odd essay of OSIS milestones in time began to converge with work of the LMNL group. In time, the name also came to mean

```
Canonical
LMNL
In
XML
```

LMNL (the Layered Markup and Annotation Language -- see [Tenn02]) overcomes the limitations of marking up non-hierarchies in XML, except that it steps entirely outside the XML paradigm to do so. It can represent unlimited overlap of all kinds. In principle LMNL is a data model, not a syntax with an implied data model as SGML was, and as XML was until DOM and the Infoset added models.

LMNL does provide a syntax, which is similar to MECS in that it does not require pure nesting like well-formed XML. Components are allowed to overlap arbitrarily, except that self-overlap requires co-indexing (as seems inherantly necessary for a flat syntax to support structures such as overlap and DAGs, that unlike trees cannot be traversed with merely a stack). For example, a tiny LMNL document could be:

```
  [paper
    [authors
      [author}me{]
```

```
        [author}you{]
      ]}...
    {paper]
```

I find this syntax superior to XML for representing arbitrary ovelap, in part because the simple (non-self-overlap) cases do not require co-indexing. However, it has the disadvantage of requiring a non-XML parser, and cannot be processed even for simple tasks by generic XML software.

Thus, I propose the following as an XML syntax for LMNL. It assures that every LMNL document expressed in it is also well-formed XML; and a certain amount of validation can be done by typical XML validators.

This syntax is "canonical" in the sense that a given LMNL document has only one corresponding CLIX form, and in the sense that it wears pointy hats; but not in the sense that it is somehow "approved" by the LMNL community, which it is not.

Creating CLIX from LMNL

Consider the base content text stream of a LMNL document as the starting point. The content characters of a LMNL document are fully ordered, so can be written out as a simple stream or file. That done, the rest of creating a CLIX document happens by inserting tags into that stream or file.

To express the rest of a LMNL document in CLIX, iterate over every range and

1. generate a unique identifier string (such as a decimal integer)
2. insert a start milestone at the start-point of the range, with that identifier as the value of its sID attribute
3. insert an end milestone at the end-point of the range, with that identifier as the value of its eID attribute

The element type for each LMNL range is equal to the type of the LMNL range, except that any character which is not in the XML name character set (such as blank, pointy brackets, etc) is transformed to a string of hex characters (0-9 and A-F) to represent the disallowed characters, using the UTF-8 encoding. Wherever one or more such hex strings occur, they must be preceded and followed by a hyphen. The hyphen character itself, if it occurs in the type name, must be represented the same way: "-2D-".

This simple procedure will create a CLIX document, which is well-formed XML, from any LMNL document.

On ordering identical ranges

In the list of ranges, two or more may be found that subsume exactly the same range of content. This cannot happen in XML, because the tags for one range must be fully contained within the other; they cannot be truly co-terminous. This property of LMNL poses a new question: can two such ranges be ordered relative to each other? We refer to such cases using one of the few "hy" words left undefined by HyTime [ISO 10744]: "HyLas". HyLas is either the *HyTime Scottish Female* form (in OSIS, this occurs only as part of the <salutation> element), or the *HyTime Locations Are the Same* structure used here.

As currently defined by the LMNL model, the relative order of HyLas ranges (ones that have the same endpoints) is undefined. More precisely, they are defined to have no ordering. Likewise, so far the order of adjacent CLIX (that is, XML empty tags) within a CLIX document is defined to not be meanginful. Thus, the CLIX equivalent of

```
    <B><I>...</I></B>
```

can be any of these:

```
    <B sID='1'/><I sID='2'/>...</I eID='2'/></B eID='1'/>
    <B sID='1'/><I sID='2'/>...</B eID='1'/></I eID='2'/>
    <I sID='2'/><B sID='1'/>...</I eID='2'/></B eID='1'/>
    <I sID='2'/><B sID='1'/>...</B eID='1'/></I eID='2'/>
```

This is the opposite of the SGML and XML model, where the linear ordering of content and markup is by definition meaningful: at the XML level the cases just shown are defined to be distinct (although applications are not required to make use of the distinction).

Which interpretation is better? Each may be, though in difference cases.

First, a simple case where the order of nesting is irrelevant to HyLAS:

```
<b><i>hello</i></b>
<i><b>hello</b></i>
```

HyLas is focused only on appearance, and the result is the same regardless of the markup’s ordering: italic/bold is the same as bold/italic. For this case, the model with no information about relative order of co-located component boundaries is enough.

Note, though, that this case depends on details of how the schema was designed. A schema might not treat bold and italic via separate elements types, but use <bi>, or (as TEI) <hi rend="bold italic">. The problem here is really that <b> and <i> are questionable tags: it is almost certain some author or encoder was trying to express some conceptual component by *using* these tags -- most likely, only one such component is present, and there are only two elements because the schema didn’t provide (or the encoder didn’t use) a portmanteau element, or a less format-oriented element.

However, there are also examples involving more substantive relationships such as when multiple TEI tags such as foreign, abbr, name, and sic share the same scope. Some cases still have relatively simple semantics, for which inheritance-based models such as (Renear et al.) merely accumulate properties: the order in which the properties are accumulated is irrelevant to the final set achieved.

But the following case suggests that sometimes HyLas must maintain order, and does not always have such simple inheritance semantics:

- Quote/Para -- It may be significant whether a quote contains a paragraph, or vice-verse; and either case might conceivably occur.
- rdgGrp/rdg, list/item, etc. -- Any homogeneous container element (that is, one which contains repetitions of a fixed pattern of sub-elements) makes sense only as the container of those elements. But since we in general allow overlap, it must be *syntactically* possible to have: <rdg><rdgGrp>...</rdg>...<rdg>...</rdg>...<rdg>...</rdgGrp></rdg>. This seems nonsensical, and providing a formal semantics for it seems a fool's errand.

Certainly there are many cases where an XML schema only allows containment in a certain order. To achieve schema validity for those cases a LMNL representation must retain order (or be schema-aware when converting to XML), so as to issue co-located element boundaries in the order that works for the particular schema. This is a case of "the things that matter, not mattering" [Usdi03]. But there are also cases where different orderings should be permitted, *and* where the differences have different meanings.

Another case for retaining order is co-occurrence of multiple empty elements (the concept of point or event semantics, not merely the XML syntactic convention or the author's SGML equivalent known as the "stupid NET trick"). For example, if several graphics, links from point anchors, or other components co-occur, their order cannot be retained currently in LMNL, though surely it may be significant in the document:

```
<img src="waterhouse.jpg"/><img src="hunt.jpg"/>
```

Another case that argues for retaining order involves the need to surround empty elements with other markup. If empty elements are treated merely as the special case where the startpoint and endpoint are equal, then other markup may refer to the same range -- but under LMNL's current semantics there is no way to keep that ordering, since the contained component was by nature empty and could not validly contain its container. A trivial case is an HTML image reference within an <a> element to make it a link.

To solve all these problems a single but substantial change to the LMNL model suffices: it must be possible to refer to markup, not merely to content. Put another way, markup must exist in nameable locations ("in-band") rather than in a separate, ineffable space ("out-of-band").

One simple model for this is to count each range start and range end location as if it were a character (or other atomic unit) of the model. They can then be referred to just like any other locations, and yet there is (as desired) no way to refer to places within the range start and end themselves -- places which may exist in syntactic realizations, but do not exist in the LMNL model itself. Another way, perhaps more mathematically clean, is to name each endpoint of each range (the CLIX sId/eID value plus an indication of start vs. end suffices), and then aggegate the endpoints for each content location into a partially-ordered set [Liu77].

Ranges co-terminous at only one end

Since ranges need not nest, but may overlap arbitrarily, there are particular cases where two ranges may have one end co-located, but the other end distinct. One might assume that the ordering at the co-located end can be set to whatever makes the ranges nest (XML-fashion); but there seems to be no ground for that assumption except its familiarity from XML. In principle, there can be cases where the opposite ordering at the co-located end may be information the model must preserve. The same examples given above can plausibly occur in this "semi-co-located" form.

The reader is cautioned to remember that "location" must still be construed as a property of the LMNL model, not of CLIX or any other syntactic realization of the model. For CLIX the mapping to syntax is trivial because every click has an ID already, but the notions are still distinct.

Range ordering with and without co-located endpoints

It is well-known that there are 13 possible ordering relationships between two ranges A and B, if overlap and identical endpoint locations are allowed. This is an unwieldy fact; there seem to be no natural-language names appropriate for many of the cases, and it is non-trivial to enumerate them all correctly, as (hopefully) done here:

B starts before A

```
1:             <a>..........</a>
    <b>...</b>

2:             <a>..........</a>
    <b>........</b>

3:             <a>..........</a>
    <b>.............</b>

4:             <a>..........</a>
    <b>.....................</b>

5:             <a>..........</a>
    <b>...........................</b>
```

B starts where A starts

```
6:     <a>...........</a>
       <b>...</b>

7:     <a>...........</a>
       <b>...........</b>

8:     <a>...........</a>
       <b>.................</b>
```

B starts within A

```
9:     <a>..............</a>
            <b>...</b>

10:    <a>..............</a>
            <b>.........</b>

11:    <a>..............</a>
            <b>...............</b>
```

B starts where A ends

```
12:    <a>...</a>
             <b>...</b>
```

B starts after A

```
13:    <a>...</a>
                    <b>...</b>
```

If empty elements are considered, two more arguably distinct cases arise:

```
14:    <a>.....</a>
       <b/>

15:    <a>.....</a>
               <b/>
```

While discussing these cases, Patrick Durusau and I noticed that 9 of the 15 cases involve co-located boundaries (cases 2, 4, 6, 7, 8, 10, 12, 14, and 15), of which case 7 is simple equality. Four of the remaining 6 cases are the well-understood basic tree-relations:

1: B precedes A

5: B contains A

9: B occupies (is contained by) A

13: B follows A

The last two cases are the overlap cases, and are symmetrical in the same way precedence and containment are. I propose the following names for them:

3: B semi-precedes A

11: B semi-follows A

Reducing the number of possible relations holding between any 2 (contiguous) ranges from 15 to 6, most of which have ubiquitous usage, seems to me a strong reason to favor LMNL adopting ordering for endpoints, rather than allowing them to be co-located.

Canonical ordering in CLIX

It should be obvious that any SGML, XML, MECS, JITTs, or LMNL document can be expressed in CLIX syntax, and that XML can be easily transformed to CLIX using XSLT. It may not be so obvious, but it is also true that this representation is amenable to storage and optimization in relational databases.

However, the model has not taken all its lix yet -- it ought not call itself "canonical" if a single LMNL document can be written out in multiple forms. The following additional rules remove the possible variations. Using them to express two LMNL documents in CLIX, means that a trivial comparison of the CLIX streams correctly determines whether the LMNL documents are equal:

1. Ranges shall be written (or at least, produce the same result as if written) in click order, which is determined by the following comparisonss (from highest order to lowest):
    1. By position of the start-points
    2. By position of the end-points
    3. By naive sort-ordering of the UCS-2 representation of the range-types
    4. Start-click, then end-click
2. Whenever a start click is written out, it is placed to follow any clicks previously written for the same location in the text content.
3. Whenever an end click is written out, it is written to precede any clicks previously written for the same location in the text content, except that the end-click for an empty range shall always immediately follow its corresponding start-click.
4. The sID values used shall be consecutive positive decimal integers assigned in order of ranges as defined above (obviously eIDs must be assigned to match). They shall be written using the Latin-1 digit characters, with no leading zeroes.
5. The five predefined XML character entities must be used in place of the corresponding characters every time those characters occur in content or in attribute values in the CLIX file.
6. Unicode surrogate characters shall be normalized according to the rules provided by the Unicode consortium. The entire file shall be written in the UCS-2 encoding.
7. No superfluous whitespace shall be inserted; the only whitespace in the CLIX file shall be whitespace that was part of the LMNL data.

Creating better XML from CLIX

To make the XML result easier to use with generic XML tools, it may be better to write many components as (non-milestoned) XML elements, rather than as milestone pairs. This makes better use of XML software's features such as property and style inheritance, scope highlighting, and so on. On the other hand, it may complicate otherwise simple click processing because not everything is just a click.

In attempting to do this well, several problems arise. Clearly, if the argument that ranges may have significant order despite having identical boundaries holds, then it is not generally possible to tell what order to write start clix in until the corresponding end clix are compared.

To ensure well-formedness:

```
Repeat for each content location in order {
   S[] = all LMNL ranges that start here
   Sort S by corresponding end locations, from first to last
   For each range Sn in S {
      Write out the XML representation for the start of Sn
      Push Sn onto open-element stack
   }
}
```

Ends are handled similarly, by inserting code like this within the outer loop above:

```
   E[] = all LMNL ranges that end here
   Sort E by corresponding start locations, from last to first
   For each range En in E {
      If En is at the top of the open-element stack {
         Write out the XML representation for the end of En
         Convert En and the corresponding start to a regular element
         Pop the open-element stack
      }
      else {
         Write out the XML representation for the end of En
         Remove (not pop) En from the open-element stack
      }
   }
```

This should always write a well-formed XML file, but it takes no account of cases where ordering matters. In general, such cases require schema-specific rules: "rdg must always be within rdgGrp", etc. Such rules must be applied first, with an algorithm similar to the above mopping up the remains. Presently, however, there is no language for expressing such rules. Some could be extracted mechanically from a schema, but it is unlikely that all cases where order must be preserved will be cases where the schema requires a particular order (again per [Usdi03], if the schema requires a particular order that is evidence that the order is not in itself meaningful).

Wendell Piez suggested a variation on CLIX that improves the ability to process it with XSLT, and provides some ability to constrain exactly what elements may cross what other elements' boundaries: While retaining the same basic name, milestoned elements could be placed in a separate namespace. This would make them distinct to an XML validator, and so (for example) milestones:quote could be allowed only in certain contexts, rather than all and only the contexts where regular quote is allowed.

§ Implementations

I have developed a checker on top of SAXON, which tests the milestone validity conditions that a generic XML validator cannot, and a full-text inverted indexer that can index both elements and Trojan milestones. Wendell Piez has created a convertor from reified LMNL to CLIX, and several interesting XSLT applications using the CLIX result to process overlapping markup phenomena. In addition, Trojan milestones are seeing rapidly-growing use due to their inclusion in OSIS, and a variety of software for Biblical applications has been or is being updated to handle this syntax and structure. Simultaneously, Bibles and related documents in countless languages are being converted into OSIS from a variety of legacy formats, for ease of future use, exchange, and archiving.

§ Summary

Overlap is a phenomenon of increasing interest for the markup community, and many approaches to representing it have been proposed. OSIS users have particularly frequent need to express overlapping structures. Among the representations that are XML-compatible, milestones have many advantages.

I propose a syntax known as "Trojan milestones", which is highly readable, easy to learn, and makes milestone forms of elements always permit the right attributes. In short, it uses the same element type both as a normal element, and as empty start- and end-milestones. This maximizes consistency between overlap and non-overlap cases, such as attribute declarations, GIs, and so on.

I also develop the use of Trojan milestones to represent LMNL documents in XML, a syntax I call "CLIX". And I discuss the problems of ordering for co-terminous LMNL ranges.

Acknowledgements

I wish to thank Wendell Piez and Patrick Durusau for many helpful comments and discussions regarding the ideas in this paper, and Troy Griffitts for the idea of Trojan milestones.

Bibliography

[Barn95] David Barnard, Lou Burnard, Jean-Pierre Gaspart, Lynne A. Price, Michael Sperberg-McQueen, and Giovanni Battista Varile. "Hierarchical Encoding of Text: Technical Problems and SGML Solutions." In The Text Encoding Initiative: Background and Contents. Guest Editors: Nancy Ide and Jean Veronis. Computers and the Humanities 29/3 (1995), pages 211-231. http://xml.coverpages.org/bib-ab.html#barnardHierarchicalCHUM

[Bray98] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. W3C Recommendation 10-February-1998.

[Clar01] James Clark and MURATA Makoto. RELAX NG Specification: Committee Specification 3 December 2001. http://www.oasis-open.org/committees/relax-ng/spec.html

[Cover] Robin Cover. "Markup Languages and (Non-) Hierarchies". http://xml.coverpages.org/hierarchies.html

[DeRo90] Steven J. DeRose, David G. Durand, Elli Mylonas, and Allen H. Renear. "What is Text, Really?" *Journal of Computing in Higher Education* 1(2): 3-26.

[DeRo93] Steven J. DeRose and David G. Durand. "Making Hypermedia Work: A User's Guide to HyTime." Kluwer Academic Publishers, 1993.

[Dura96] David G. Durand, Elli Mylonas, Steven J. DeRose. "What Should Markup Really Be? Applying theories of text to the design of markup systems." Presented at The Joint International Conference: ALLC/ACH 1996. http://xml.coverpages.org/DurandWhatShouldTextBe-ALLC1996.pdf

[Duru02a] Patrick Durusau and Matthew Brook O'Donnell."Coming down from the trees: Next step in the evolution of markup?" Late breaking paper presented at Extreme Markup, Montreal, 2002.

[Duru02b] Patrick Durusau and Matthew Brook O'Donnell."Concurrent Markup for XML Documents." Presented at XML Europe 2002. http://www.idealliance.org/papers/xmle02/dx_xmle02/papers/03-03-07/03-03-07.html

[Duru03] Patrick Durusau and Steven J. DeRose. "OSIS: A Users' Guide to the Open Scripture Information Standard." Bible Technologies Group, 2003.

[ISO 10744] International Organisation for Standardisation. 1992. ISO/IEC 10744. Hypermedia/Time-based Structuring Language: HyTime.

[ISO 8879] International Organization for Standardization. 1986. ISO 8879: 1986(E). Information Processing: Text and Office Information Systems: Standard Generalized Markup Language.

[Liu77] Liu, C. L. 1977. Elements of Discrete Mathematics. New York: McGraw-Hill. ISBN 0-07-038131-3.

[LMNL] --------. LMNL home page. http://www.lmnl.net

[RDF04] Resource Description Framework (RDF): Concepts and Abstract Syntax. W3C Recommendation 10 February 2004. http://www.w3.org/TR/rdf-concepts/

[Rene95] Allen Renear, Elli Mylonas, and David Durand. "Refining our Notion of What Text Really Is: The Problem of Overlapping Hierarchies." In Research in Humanities Computing, 1993/1995.

[RLMNL] --------. Reified LMNL. http://www.lmnl.net/prose/data-model/reified-LMNL.html

[Sper98] C. M. Sperberg-McQueen and Claus Huitfeldt. "Concurrent Document Hierarchies in MECS and SGML". Presented at The Joint International Conference: ALLC/ACH 1998. July 5 - 10, 1998, Lajos Kossuth University , Debrecen, Hungary. Association for Literary and Linguistic Computing; Association for Computers and the Humanities. http://lingua.arts.klte.hu/allcach98/abst/abs47.htm

[Sper99] C.M. Sperberg-McQueen and Claus Huitfeldt. "GODDAG: A Data Structure for Overlapping Hierarchies." Presented at ACH-ALLC ’99. Reprined in the Proceedings of PODDP’00 and DDEP’00, edited by Anne Bruggemann-Klein and Ethan Munson. New York: Springer, 2001. http://www.iath.virginia.edu/ach-allc.99/proceedings/sperberg-mcqueen.html&e=7781

[TEI] Michael Sperberg-McQueen and Lou Burnard (eds). Technical Topics: Multiple Hierarchies. Chapter 31 in the TEI Guidelines for Electronic Text Encoding and Interchange. http://xml.coverpages.org/teichap31.html

[TEIP4] Text Encoding Initiative. The XML Version of the TEI Guidelines. EI P4: Guidelines for Electronic Text Encoding and Interchange. The TEI Consortium. Edited by C M Sperberg-McQueen and Lou Burnard. XML conversion by Syd Bauman, Lou Burnard, Steven DeRose, and Sebastian Rahtz. http://www.tei-c.org/P4X/

[Tenn02] Jeni Tennison and Wendell Piez. "The Layered Markup and Annotation Language (LMNL)" Late breaking paper presented at Extreme Markup, Montreal, 2002.

[TM01] XML Topic Maps (XTM) 1.0: TopicMaps.Org Specification. http://www.topicmaps.org/xtm/1.0/xtm1-20010806.html

[Usdi03] B. Tommie Usdin. "When ’It Doesn’t Matter’ means ’It Matters’". Presented at Extreme Markup 2002.

[XLink01] Steven DeRose, Eve Maler, and David Orchard (eds.). XML Linking Language (XLink) Version 1.0. W3C Recommendation 27 June 2001. http://www.w3.org/TR/xlink/

[XPtr01] Paul Grosso, Eve Maler, Jonathan Marsh, Norman Walsh (eds.) XPointer element() Scheme. W3C Recommendation. 2001. http://www.w3.org/TR/xptr-element/

The Author

Steven DeRose
Bible Technologies Group
1908 Wallace Ave.
Silver Spring
MD
20902
US
tel: 301-949-6544
fax:
sderose@acm.org
http://www.derose.net

Steven DeRose has been working with electronic books and hypertext for many years, and has served on standards committees including XML, XLink, XPath, XPointer, TEI, OSIS, and others. He is a frequent speaker on linguistics, markup, and other topics, and has published two books and many articles. He presently serves as Chairman of the Bible Technology Group.

Extreme Markup Languages 2004®
Montréal, Québec, August 2-6, 2004
This paper was formatted from XML source via XSL
by Mulberry Technologies, Inc.