Hierarchical Encoding of Text: Technical Problems and SGML Solutions[1]

David T. Barnard

Queen's University

Lou Burnard

Oxford University

Jean-Pierre Gaspart

Associated Consultants and Software Engineers sa/nv

Lynne A. Price

Frame Technology Corp.

C.M. Sperberg-McQueen

University of Illinois at Chicago

Giovanni Battista Varile

Commission of the European Communities

June 1994

[1]This paper is derived from a working paper of the Metalanguage Committee entitled "Notes on SGML Solutions to Markup Problems" which was produced following a meeting of the committee in Luxembourg. The co-authors all participated in that meeting and provided input to this paper. Others serving on the committee at other times included David Durand (Boston University), Nancy Ide (Vassar College) and Frank Tompa (University of Waterloo).

1 Introduction

The straightforward way to use the Standard Generalized Markup Language (SGML)[2] is to define the content of a document as a number of components which in turn are defined by giving their components, and so on. The result of this approach is a hierarchical description of a document, in which the components are called elements. While this seems natural in many settings—a book has front matter, a body, and back matter, each of which can be described in more detail—it is too limiting for many documents of interest to participants in the Text Encoding Initiative (TEI). Many texts, or portions of texts, that are to be encoded using the TEI Guidelines can be viewed in more than one (hierarchical) way. One recurring theme in the project has been the need to represent non-hierarchical information in a natural way (or, at least, in a way that is acceptable to those who must use it) using a technical tool that assumes a single hierarchical representation.[3]

While some of the other papers in this volume address specific needs for non-hierarchical encodings, this paper explores SGML-based techniques for representation of data that is almost but not quite hierarchical in a notation for expressing a hierarchy.

Although some SGML constructs are common to more than one of the topics discussed, the sections of this paper are largely independent. The reader is assumed to be familiar with the general principles and use of SGML[4], but not to be expert on the details.

The examples were inspired by discussions of TEI's metalanguage committee, often in response to specific problems identified by other committees and working groups in the project. The committee explored alternative encodings to understand the advantages and disadvantages of each. Rather than recommending a single approach to each problem, the committee concluded that working groups and individual researchers should make informed decisions about the techniques appropriate to individual requirements. Any more restrictive decision would force some TEI users to increase the encoding effort (perhaps prohibitively) with no immediate benefit.

The coding schemes shown below represent the committee's ideas on appropriate SGML mechanisms for logical problems; they do not necessarily reflect the views of the committee concerning the individual application areas. In the examples, especially, the focus is on markup problems; the examples in this paper are not intended as full analyses. Researchers working on hypertext encodings should also consult the paper by Steven DeRose and David Durand.[5] The examples and solutions described in this document are intended to be pedagogical; divergences in this document from the TEI guidelines or the HyTime standard should not be taken as rejections of those recommendations. In particular, some names are used for tags that are used in other ways or with other attributes in the TEI Guidelines;

[2] ISO (International Organization for Standardization). *ISO 8879-1986 (E). Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML).* First Edition — 1986-10-15. [Geneva]: International Organization for Standardization, 1986. See also Lou Burnard, *What is SGML and how does it help?*, in this volume.

[3] This is not precisely true, as will become clear below. SGML does have a method for representing concurrent hierarchies, but there was concern from the beginning of the project about the applicability of the CONCUR feature.

[4] Perhaps from reading Lou Burnard's paper in this volume.

[5] Steven J. DeRose and David G. Durand. *The TEI Hypertext Guidelines.* In this volume.

these are used in this paper as examples only.

The ideas presented in this paper have been influential in the work of the TEI to date. As can be seen from other papers in this journal issue, as well as in the Guidelines themselves, the committees and work groups of the TEI have made choices among the options described here while designing the TEI encodings for different kinds of texts.

In addition to their historical and pedagogical interest, the ideas presented here can serve as a model of the local extensions permitted by the TEI Guidelines. One of the features of the TEI encoding scheme is its extensibility[6], which allows users to change existing features of the encoding scheme, or add new features to it, in controlled ways. Some of these local extensions may well involve dealing with other kinds of non-hierarchical structures than those already accommodated in the Guidelines, and users designing those extensions should find this paper to be of value.

In contrast with other papers in this issue which are about specific aspects of the content of the Guidelines, this paper is about SGML and the ways in which it might be used to accomplish some of the things accomplished in the Guidelines, or in extensions of the guidelines that others might wish to make.

2 Supplementing a Document's Hierarchy

An encoder often needs to indicate arbitrary segments whose boundaries do not reflect a document's primary hierarchy, or indeed, any hierarchical structure. This problem is illustrated by the frequent need to annotate a text with in-line comments such as the following:

- In an encoding of a particular copy of a work: "the passage is illegibly water stained".[7]
- In a transcript of an interview: "reply rendered inaudible by a passing truck".
- In a gestural transcription of a conversation: "Nino moved to the window".
- In minutes of a meeting covering tariff policy, foreign relations with Japan (including cultural exchange, economic cooperation, tariff policy, patent law), and wheat production (including a digression on wheat tariffs): "this section discusses tariffs".

The discussion of this problem is long because several SGML concepts are introduced in the discussion, and because the large number of situations that require non-hierarchical material suggests a large number of possible solutions.

[6]See Chapters 29 and, to a lesser extent, 28 of the Guidelines.

[7]Or more elaborately, the passage marked by straight lines in the left margin, the passage marked by wavy lines in the left margin, the passage underlined by hand with a simple straight line, the passage underlined by hand with a simple straight line which was later deleted by hand, etc., as in the transcriptions of Wittgenstein's manuscripts in the Norwegian Wittgenstein project. See Claus Huitfeldt and Viggo Rossvoer, The Norwegian Wittgenstein Project Report 1988 ([Bergen]: NAVFs EDB-Senter for Humanistisk Forskning / Norwegian Centre for the Humanities, 1989), especially. pp. 201-236.

2.1 Boundary Elements for Arbitrary Events

The first solution introduces special elements into the document whose sole purpose is to mark the boundaries of such passages. While most SGML elements consist of content (in general, a combination of text and other elements) delimited by a start-tag at the beginning and an end-tag at the end, other SGML elements mark points in the text, but have no content. These elements have declared content EMPTY in the DTD. They provide information about the text but do not contain any text or other elements; this information is completely contained in the generic identifier and attributes in the start-tag, and the location of the element within the document hierarchy. Such elements have start-tags, but no end-tags.

For example, let us assume a detailed gestural transcription of an interview in which the structural markup is concerned only with the speeches made, but in which a general purpose "action" element is also used to describe the actions of the participants. Suppose, for example, that during the session Nino walked to the window, stood there, and, after a few minutes lit a cigarette, returning to the table before putting it out. Using the designations "passage A", "passage B" and so on to indicate the primary hierarchy of the transcript, the "action" element might be used as follows:[8] [9]

```
<TEI.2>
...
<action start id=a237 person=nino desc='sits at table'>
  passage A
<action end id=a237>
<action start id=a238 person=nino desc='walks to window'>
  passage B
<action end actionId=a238>
<action start id=a239 person=nino desc='stands at window'>
  passage C
<action start id=a240 person=nino desc='smokes'>
  passage D
<action end actionId=a239>
<action start id=a241 person=nino desc='walks to table'>
  passage E
<action end actionId=a241>
  passage F
<action end actionId=a240>
  passage G
...
```

[8]This "action" element is similar to the "milestone" defined in the TEI Guidelines for segmentation of text according to pagination of multiple editions. However, "milestone" assumes a simple single-level segmentation of the text: the attribute values specified in any "milestone" element apply to all following text until the next "milestone" belonging to the same edition. Hence, no explicit end marker is needed and the ID/IDREF mechanism is not needed.

[9]Note that in the tag set for transcribing spoken text as actually published in P3, a different solution was adopted for this problem, involving the use of an empty "event" element synchronized by means of a mechanism similar to that described elsewhere in this article.

```
</TEI.2>
```

Here, actions have been arbitrarily numbered a1, a2, and so on. These action numbers are indicated by the value of the "id" attribute at the start of the action and the "actionId" attribute at the end. Custom application software can use this attribute to link the two ends of the action, recognizing, for example, that Nino was smoking throughout passages D-F. The application software can also assume that values for the "person" and "desc" attributes are specified at the start of the action.

The start and end of each action is indicated by the value of an "endpoint" attribute. Since the possible values of "endpoint" are predefined to be either "start" or "end", when the optional SHORTTAG feature is used, SGML allows the attribute name to be omitted from the start-tag. Thus, a tag such as:

```
<action end actionId=a241>
```

is an abbreviated form of the more complete:

```
<action endpoint=end actionId=e241>
```

The action element might be defined in SGML as follows:

```
<!ELEMENT action - O EMPTY >
<!ATTLIST action
  id       ID    #IMPLIED
  actionId IDREF #IMPLIED
  person   NAME  #IMPLIED
  desc     CDATA #IMPLIED
  endpoint (start | end) #IMPLIED >
]>
```

The element declaration for "action" does not give a content model. Instead the declared content EMPTY indicates that the element is used as described above.

Note that "id" has the declared value ID and "actionId" has the declared value IDREF. The values of these attributes are SGML unique identifiers. SGML allows every element to be associated with a string of characters called a unique identifier. Only one element in a document instance may have a given unique identifier, regardless of its generic identifier. Thus not only must every two "chapter" elements, for instance, have different unique identifiers, but also the unique identifier of a "chapter" cannot be the same as that of a "figure".

An element's unique identifier is specified as the value of an attribute whose declared value is ID. Since an element cannot have more than one unique identifier, an element can have only one ID attribute. The SGML Standard recommends that the same name be used for the ID attributes of all elements that have such attributes.

A unique identifier is an SGML name. Under the reference concrete syntax, therefore, it starts with a letter, and may contain up to eight letters, digits, hyphens and periods. These restrictions are often modified. This paper, for example, assumes that the relevant SGML declaration increases the NAMELEN quantity to allow longer names.

Elements may also have attributes that identify other elements through their unique identifiers. In particular, the value of an attribute with the declared value IDREF must be the same as that of the ID value of some element (the same one, a preceding one, or a following one); the declared value IDREFS means the value is a list of one or more ID values.

The "id" and "actionId" attributes declared above for the "action" element use this mechanism. While an SGML parser does not ensure that an "id" is specified whenever "endpoint" is "start", custom application software could do so. Custom software could also verify that the encoder honors other application conventions, such as specifying a "actionId" whenever "endpoint" is "end", specifying both a start and an end for each action, and not specifying more than one end for a started action.

At the expense of increasing the number of element types, this DTD can be modified so that SGML ensures that the right combination of attributes is specified at each end of the action. Here, instead of using a single "action" element type for both endpoints, there are separate "actionStart" and "actionEnd" element types:

```
<TEI.2>
...
<actionStart id=a237 person=nino desc='sits at table'>
  passage A
<actionEnd id=a237>
<actionStart id=a238 person=nino desc='walks to window'>
  passage B
<actionEnd actionId=e238>
<actionStart id=a239 person=nino desc='stands at window'>
  passage C
<actionStart id=a240 person=nino desc='smokes'>
  passage D.
<actionEnd actionId=e239>
<actionStart id=a241 person=nino desc='walks to table'>
  passage E
<actionEnd actionId=e241>
  passage F
<actionEnd actionId=e240>
  passage G
...
</TEI.2>
```

2.2 Typed Boundary Elements

The method just described assumes that the kinds of action to be encoded (i.e., the possible values for the "desc" attribute) will not be known until the transcription is complete. When the possible event types are known in advance, they can be defined in the DTD as different generic identifiers:

```
<!ELEMENT (move|gesture|...) - O EMPTY>
```

```
<!ATTLIST (move|gesture|...)
  id      ID    #IMPLIED
  eventid IDREF #IMPLIED
  person (nino|lynne|lou|michael|pierre) #IMPLIED
  desc    CDATA #IMPLIED
  endpoint (start | end) #IMPLIED >
```

This DTD fragment also assumes that the participants are known in advance and can be listed explicitly in the declared value for the attribute "person". This allows the attribute name to be omitted when the value is given. Nino's walk to the window would therefore be encoded as:

```
<move start id=e238 nino desc='walks to window'>...
<move end eventid=e238>
```

2.3 Concurrent Markup

SGML's optional CONCUR feature provides a different approach. Although most SGML document instances represent a single hierarchy, CONCUR allows the same data characters to be described in multiple hierarchies, presenting different views of a document. In fact, CONCUR requires that all of the data characters be incorporated into each view. There is a separate DTD for each one; the first is known as the base document type declaration. When CONCUR is used, the generic identifier in a start-tag or an end-tag can be preceded by one or more document type names, separated by a connector and enclosed in group delimiters, to indicate that the tag applies only to the named DTDs. When a start-tag does not contain a document type name, it applies to all active views. For example, meeting minutes might be transcribed using one DTD for the text of speeches, and separate DTDs for the gestures of each participant. The DTD for Nino's possible movements might include declarations such as:

```
<!DOCTYPE nino [
<!ELEMENT nino (#PCDATA | move | gesture |...)+ >
<!ELEMENT move (#PCDATA) >
<!ELEMENT gesture (#PCDATA) >
<!ATTLIST (move, gesture,...) desc CDATA #IMPLIED>
]>
```

so that Nino's move to the window is bounded by:

```
<(nino)move desc='walks to window'>...</(nino)move>
```

where any speech or other gestures Nino made while he was walking are represented by the ellipsis.

This encoding assumes that no two gestures, movements, or other events can overlap or nest. If events may overlap or occur within each other, the illustrated element declarations must be revised appropriately to describe a suitable hierarchy. For example,

```
<!ELEMENT move (#PCDATA|gesture)+ >
<!ELEMENT gesture (#PCDATA) >
```

indicates that gestures may occur within movements, but does not allow the reverse. If there are no restrictions on the combinations of elements that may occur within each other, the gestural tags may all be defined as inclusion exceptions on the document type element and each other:

```
<!ELEMENT nino                - O
      (#PCDATA) +(move | gesture |...) >
<!ELEMENT (move, gesture,...) - O
      (#PCDATA) +(move | gesture |...) >
```

Legibility or audibility information can be readily accommodated by CONCUR with a "leg" or "aud" DTD segmenting a document into segments of a given legibility or audibility. The DTD might look something like this:

```
<!DOCTYPE aud [
<!ELEMENT aud - - (clear | distorted | inaudible)+ >
<!ELEMENT clear - - (#PCDATA) >
<!ELEMENT distorted - - (#PCDATA) >
<!ELEMENT inaudible - - (#PCDATA) >
<!ATTLIST (distorted, inaudible) cause CDATA #IMPLIED >
]>
```

Here, the content model for "aud" requires every portion of a document to be coded as clear, distorted, or inaudible; the content models for "clear", "distorted", and "inaudible" prevent a single portion from belonging to more than one of these sections. A variation would provide a "DNA" option to be used when audibility does not apply (for example, to headings and annotations). Thus, interspersed among other markup, the encoded document could contain sequences such as:

```
<(aud)clear>...</(aud)clear>
<(aud)inaudible
cause='truck'>...</(aud)inaudible>
<(aud)clear>...</(aud)clear>
<(aud)distorted
cause='volume overload'>...</(aud)distorted>
<(aud)clear>...</(aud)clear>
```

There are limitations to this approach. First, it is necessary to define a different DTD for each type of coding (for example, for each participant in a conversation). Second, SGML parsers need not support the CONCUR feature and most currently available parsers do not. Third, there is the burden of additional markup to be inserted in the document; unfortunately the prefixing of tag names by DTD names is required for all tag names that do not occur in all the DTDs being concurrently processed, so that introducing a new concurrent variant can require adding a prefix to all existing tag names in a document.

2.4 Separate Analysis Elements

Even when CONCUR is not available, coding of independent passages does not need to be embedded in the existing hierarchy. In this next variation, the primary hierarchy contains the text of the transcription, indicating the boundaries of the desired segments. Relationships among these segments are indicated in a separate "analysis" element, which follows.[10] Where existing elements of the main hierarchy do not precisely define the desired segment, special segmentation elements (with the generic identifier "s") provide additional markers.

Using this method, the document instance has three sections. The first is the transcription of the meeting, showing the segmentation of the text. Comments have been added to make the markup clear, but the comments are totally redundant and any processing software ignores them. More detail is shown here than in previous examples. In particular, the text is divided into paragraphs, which in turn are divided into segments to show the start and end of gestures as needed. When the paragraph boundaries do not coincide with the passages identified between gestures above, the passages are split into pieces:

```
<TEI.2>
...
<p>...
  <s id=s101><!--Nino sits at table-->passage A</s>
  <s id=s102><!--Nino walks to window-->passage B1</s>
</p>
<p>
  <s id=s103>passage B2<!--Nino reaches the window--></s>
  <s id=s104>
    <!-- Nino stands at window -->passage C
    <s id=s105><!-- Nino starts smoking -->passage D1</s>
  </s>
</p>
<p>
  passage D2...
  <s id=s106><!--Nino walks to table-->passage E</s>
  <s id=s107><!-- Nino arrives at table-->passage F1
</p>
<p id=p235>passage F2</p>
<p id=p238>passage G</p>
...
</TEI.2>
```

In a separate section we define the gestures detected during analysis. A number of possibilities exist for fopr the representation of such abstract structures, many of which are described in chapters 14 to 16 of P3. In this example we use a simplified version of the "feature structure" analysis discussed in chapter 16 of P3. The elements "fStruct" and "feature" used here correspond with the "fs" and "f" elements discussed in that chapter.

[10]This is essentially how most of the analytic elements discussed in Chapters 14 and 15 of P3 work.

Here, the ordering of "fStruct" elements is irrelevant, because these gestures have not yet been related to the sequence of events marked in the text:

```
<analysis>
  <fStruct id=m1>
    <feature name=who>Nino</feature>
    <feature name=act>sits</feature>
    <feature name=loc>at table</feature>
  </fStruct>
  <fStruct id=m2>
    <feature name=who>Nino</feature>
    <feature name=act>walks</feature>
    <feature name=loc>table to window</feature>
  </fStruct>
  <fStruct id=m3>
    <feature name=who>Nino</feature>
    <feature name=act>stands</feature>
    <feature name=loc>at window</feature>
  </fStruct>
  <fStruct id=m4>
    <feature name=who>Nino</feature>
    <feature name=act>smokes</feature
  </fStruct>
  <fStruct id=m6>
    <feature name=who>Nino</feature>
    <feature name=act>walks</feature>
    <feature name=loc>window to table</feature>
  </fStruct>
</analysis>
```

The final section of the document encodes the relationships between the first two. Using IDREF attributes it connects the analyzed gestures with the actual text, using the unique identifiers that were assigned to the gestures in the analysis, and to the paragraphs and segments in the text:

```
<TEI.2>
...
  <alMap><!-- Nino sitting at table, passage A -->
    <alPtr target=m1><alRange alStart=s101 alEnd=s102>
  </alMap>
  <alMap><!-- Nino walking to window, passage B-->
    <alPtr target=m2><alRange alStart=s102 alEnd=s103>
    <!-- Note that Nino stops sitting as he begins walking.-->
  </alMap>
  <alMap><!-- Nino at window, passages C, D -->
```

```
    <alPtr target=m3><alRange alStart=s104 alEnd=s106>
  </alMap>
  <alMap><!-- Nino smokes, passages D-F -->
    <alPtr target=m4><alRange alStart=s105 alEnd=p238>
  </alMap>
  <alMap><!-- Nino walks to table, passage E -->
    <alPtr target=m6><alPtr alStart=s106 alEnd=s107>
  </alMap>
  <alMap><!-- Nino sitting at table, passage G-->
    <alPtr target=m1><alRange alStart=p235 alEnd=p238>
    <!-- Note: fStruct from passage A reused-->
  </alMap>
</alignment>
...
</TEI.2>
```

Although in this example, all the recorded events have separate start and end points, this approach applies equally well to instantaneous events that have a single time point. The "alPtr" element can provide the unique identifier of an element in the text as well as of an element in the analysis. Alignment mappings such as the following are then possible:

```
<alMap>
  <alPtr target=m17><alPtr target=p726>
</alMap>
```

This approach provides a more declarative interpretation of non-hierarchical material; it also automatically provides for discontinuous segments (such as sitting at the table, in this example). Its obvious disadvantage is the bulk and complexity of the markup.

In practice, subsequent work in the two groups responsible for linguistic analysis and hypertext resulted in a reasonably simple and more general mechanism for the encoding of this kind of standalone linking, using the same basic notion of an external element used to align text segments and their analyses. Other papers in this volume (notably those by Durand and DeRose, and Langendoen and Simons) and the relevant chapters of the Guidelines should be consulted for more detail.

2.5 Discussion

The relevance of these solutions to different situations depends on a variety of factors. The effort of encoding an analysis such as that in Section 1.4 is likely to be prohibitive if the primary purpose of the coding effort is to record the hierarchy of the text rather than the non-hierarchical material, but is to be expected if the focus of the research is the latter. How many types of annotations are to be made? Are the classifications known in advance? The answers to these questions help determine whether a general "event" tag or specific "gesture", "move", "walk", "set", elements are appropriate. Other considerations include the availability of CONCUR and the ease of developing custom applications. Users should choose one of the methods described above, or a variation, depending on specific circumstances.

3 Relating Non-adjacent Text

Another problem of a non-hierarchic nature involves encoding relationships among nonadjacent segments of text.[11] Examples include:

- words rendered illegible by the stain on the right-hand side of a page,
- the finite verb "stellte vor" in the German sentence "Er stellte seine These den Kollegen hoffnungsvoll vor",
- the discussion of tariffs in parliamentary minutes (assuming that the discussion wanders back and forth from one topic to another), and
- the root KTB in the Arabic word "al-kaatib".

3.1 Labeling Related Passages with Unique Identifiers

The first solution uses a single SGML unique identifier to label all related passages. Most occurrences of this unique identifier appear as the value of an IDREF attribute, since it can be an ID attribute only once. When the nonadjacent passages are elements of the document's natural hierarchy, the coding simply involves adding the attributes to the start-tags that would be used in any case. In the following sample, a topic is given an ID and a full description the first time it occurs; thereafter, an IDREF attribute refers to the earlier labelling:

```
<!DOCTYPE TEI.2 system "tei2.dtd" [
  ...
  <!ELEMENT topic - - (#PCDATA) >
  <!ATTLIST topic
    id      ID    #IMPLIED
    full    CDATA #IMPLIED
    topicid IDREF #IMPLIED
  >
]>
<TEI.2>
...
<topic id=tariff full='Tariffs on steel'>...</topic>
<topic id=wheat full='Wheat Crop Projections'>...</topic>
<topic topicid=tariff>...</topic>
<topic id=flag full='National Flag Month'>...</topic>
<topic topicid=tariff>...</topic>...
...
</TEI.2>
```

[11] An equivalent facility is provided in P3 by the "span" element.

This coding artificially distinguishes the first occurrence of a topic from any others. Furthermore, since all attributes are optional, SGML does not enforce the intended use of the attributes. Both limitations can be avoided by defining a topic list as part of the document's front matter. This register contains the ID attributes; the actual text contains IDREF attributes:

```
<!DOCTYPE TEI.2 system "tei2.dtd" [
  ...
  <!ELEMENT topic_declaration - O EMPTY >
  <!ATTLIST topic_declaration
    full CDATA #REQUIRED
    ID   ID    #REQUIRED
  >
  <!ELEMENT topic - - (#PCDATA) >
  <!ATTLIST topic topicid IDREF #REQUIRED >
]>
<TEI.2>
...
<front>
  ...
  <topiclist>
    <topic_declaration id='tariff' full='Tariffs on steel'>
    <topic_declaration id='wheat' full='Wheat Crop Projections'>
    <topic_declaration id='flag' full='National Flag Month'>...
  </topiclist>
  ...
</front>
<body>
  ...
  <topic topicid='tariff'>...</topic>
  <topic topicid='wheat'>...</topic>
  <topic topicid='tariff'>...</topic>
  <topic topicid='flag'>...</topic>
  <topic topicid='tariff'>...</topic>
  ...
</body>
</text>
</TEI.2>
```

Unique identifiers can also be used to indicate coincidental relationships determined by presentation rather than textual content. Here, CONCUR is used to identify the nonadjacent passages. CONCUR works well with unique identifiers, especially since an IDREF attribute must refer to a unique identifier defined in the same document instance. Thus, when CONCUR is being used, an IDREF attribute cannot refer to an element in a different view. For example, using "leg" to mean "legibility", "s" for "stain", "i" for "ink blot", "t" for “torn”, suppose the purpose of the encoding is to indicate passages’ legibility. The new DTD might be as follows:

```
<!DOCTYPE leg [
<!ELEMENT leg (#PCDATA | s | i | t)+ >
<!ELEMENT s (#PCDATA) >
<!ATTLIST (s, i, t)
  id ID #IMPLIED
   segid IDREF #IMPLIED >
]>
```

If “rhs.p23” is the unique identifier indicating “right-hand side of page 23”, a typical document sequence might be:

> ... Random statistical quirk for the day: the word ”no” appears 1344 times in the King James Bible, but the ¡(leg)s id=rhs.p23¿word¡/(leg)s¿ ”yes” appears only twice! (Grep for¡(leg)s segid=rhs.p23¿ yourself if yo¡/(leg)s¿u don’t believe me). At¡(leg)s segid=rhs.p23¿ first I thought thi¡/(leg)s¿s was just a hilarious¡(leg)s segid=rhs.p23¿ artifact of religious¡/(leg)s¿ dogma, so I chec¡(leg)s segid=rhs.p23¿ked Alice in Wonderlan¡/(leg)s¿d – ”yes” appears onl¡(leg)s segid=rhs.p23¿y once! Curiouser ¡/(leg)s¿and curiouser. Well it¡(leg)s segid=rhs.p23¿ turns out to be¡/(leg)s¿ a property of English (yes¡(leg)s segid=rhs.p23¿/no = .0¡/(leg)s¿66 on average), and when you consider why this might be, it’s undoubtedly due to the fact...[12]

A register of stains, similar to the above register of topics, can be used to provide more symmetric markup.

3.2 Repeating Related Material

A second solution is simply to gather the related segments in one place and repeat them. This solution is appropriate for short passages, such as relationships among discontiguous portions of a word. Thus the Arabic root mentioned above might be encoded as:

```
<word root=KTB>al-kaatib</word>
```

or

```
<word>
  <root>KTB</root>
  <form>al-kaatib</form>
</word>
```

The first variation emphasizes that “KTB” is the result of analysis rather than part of the text being lemmatized.

[12] Humanist 3.769, Tue, 21 Nov 89, posting from mike@tome.media.mit.edu (Michael Hawley).

3.3 Aligning Non-adjacent Passages

The alignment map illustrated under Section 2.4 can also be applied to the problems described in this section.

4 Handling Ambiguous Content

The next problem addressed is that of encoding multiple analyses of the same content. Examples include:

- Multiple parsings of the gross syntactic structure of the sentence "I saw the man with the telescope"
- The pagination of various editions of Shakespeare's Hamlet

4.1 Concurrent Markup

The first solution uses CONCUR. Each encoding (for the above examples, each edition, or each parse) is assigned a different DTD. Except for the declaration of the document element, these DTDs are essentially equivalent. Suppose the different editions of the play, for instance, are to be represented as a sequence of "page" elements. Within each "page", the "omitted" element is used to indicate text that does not occur in the indicated edition. Using "F" to refer to the First Folio, "Q1" to the First Quarto, "Q2" to the Second Quarto, and "Ri" to the Riverside Shakespeare edition, the document instance set might be coded as follows:

```
<!--Start document instance in all views:-->
<(TEI.2)TEI.2><(f)f><(q1)q1><(q2)q2><(Ri)Ri>

<!-- Set up TEI encoding in the (TEI.2) view only -->
<(f)omitted><(q1)omitted><(q2)omitted><(Ri)omitted>
<(TEI.2)teiHeader>...</(TEI.2)teiHeader>
</(f)omitted></(q1)omitted></(q2)omitted></(Ri)omitted>
<(TEI.2)text><(TEI.2)body>
<!-- Act 1, Scene 1 starts... -->
<(TEI.2)div1 name='act' n='1'>

<!-- initial pagination for various editions -->
<(F)page n='g5a'>
<(Q1)page n='3'>
<(Q2)page n='[3]'>
<(Ri)page n='234'>

<!--Start text and indicate new pages as they occur -->
... text of Hamlet...
</(F)page><(F)page n='g5b'>
```

```
... text of Hamlet...
</(Q2)page><(Q2)page n='4'>
... text of Hamlet...
</(Ri)page><(Ri)page n='235'>
... text of Hamlet...
</(F)page><(F)page n='g5b'>
... text of Hamlet (to end)
</(f)page></(q1)page></(q2)page></(Ri)page>
</(TEI.2)body>
</(TEI.2)text>
</(TEI.2)TEI.2>
```

The different DTDs can easily be defined using a common entity. For example, suppose the system identifier "page.dec" refers to a file with the declarations pertaining to "page", "omitted", and related elements. In addition this file can contain a declaration such as:

```
<!ELEMENT %version.name - - (page)* +(omitted) >
```

To define the individual DTDs, the prolog of the SGML document would repeatedly refer to this entity:

```
<!DOCTYPE F system "page.dec" [<!ENTITY % version.name "F" >]>
<!DOCTYPE Q1 system "page.dec" [<!ENTITY % version.name "Q1" >]>
<!DOCTYPE Q2 system "page.dec" [<!ENTITY % version.name "Q2" >]>
<!DOCTYPE Ri system "page.dec" [<!ENTITY % version.name "Ri" >]>
```

A similar technique can be applied to ambiguous sentences. Here, however, there are no obvious meaningful names to assign the different DTDs; they can simply be called "a", "b", "c" and so on. One flaw in this method is that it limits the number of parses that can be encoded for a sentence to the number of DTDs that are defined. Furthermore, although the markup suggests a relationship between all components of one view, if a text contains more than one ambiguous sentence, there need be no special relationship between the particular parses coded under a particular DTD.[13].

4.2 Repeated Content

Another way to represent the different interpretations is to repeat the ambiguous material, encoding a different analysis with each repetition. For example, the ambiguous sentence quoted above could be represented as:

```
<sentence>
  <parse>
    <s>
      <np>I</np>
      <vp>
```

[13]This method is further discussed in chapter 31 of P3

```
        <vp>
          <v>saw</v>
          <np>the man</np>
        </vp>
        <pp>
          <p>with</p>
          <np>the telescope</np>
        </pp>
      </vp>
    </s>
  </parse>
  <parse>
    <s>
      <np>I</np>
      <vp>
        <v>saw</v>
        <np>
          <np>the man</np>
          <pp>
            <p>with</p>
            <np>the telescope</np>
          </pp>
        </np>
      </vp>
    </s>
  </parse>
</sentence>
```

This method works well unless CONCUR is used and the ambiguity is not common to all views. Since the repeated content is present in every view, this coding changes the content of views that do not have the ambiguity. In addition, of course, the encoding does not equate corresponding units (words, for example) of the different repetitions.

4.3 Encoding Application-Specific Analyses

An element structure particular to the application can often be defined. The next encoding is based on a chart representation of two parses. In a chart, the words or tokens in a sentence are arranged on a line; arcs with endpoints between the words on this line represent the phrases formed by the words between their endpoints. The arcs are labelled with a marker indicating the phrase type.

One simple method of representing a chart in SGML is to number the words in the sentence. The points between words are then also numbered: the start of the sentence is 0 and the point after each word has the same number as the preceding word. EMPTY elements with attributes identifying the phrase marker and the endpoints then represent the arcs. The DTD might be:

```
<!DOCTYPE sentence [
<!ELEMENT sentence (text, parse*) >
<!ELEMENT text (#PCDATA) >
<!ELEMENT parse (arc)+ >
<!ELEMENT arc EMPTY >

<!ATTLIST arc marker (s, np, vp, pp, v, n, p) #IMPLIED
              s NUMBER #REQUIRED --start of arc--
              e NUMBER #REQUIRED --end of arc-- >
]>
```

The sample sentence with its two analyses is then represented as:

```
<sentence>
<text>I saw the man with the telescope</text>
<parse>
<arc S s=0 e=7>
<arc NP s=0 e=1>
<arc VP s=1 e=7>
<arc VP s=1 e=4> <!-- saw the man -->
<arc V s=1 e=2> <!-- saw -->
<arc NP s=2 e=4> <!-- the man -->
<arc PP s=4 e=7> <!-- with the telescope -->
<arc P s=4 e=5> <!-- with -->
<arc NP s=5 e=7> <!-- the telescope -->
</parse>
<parse>
<arc S s=0 e=7>
<arc NP s=0 e=1> <!-- I -->
<arc VP s=1 e=7>
<arc V s=1 e=2> <!-- saw -->
<arc NP s=2 e=7> <!-- the man w/ the telescope -->
<arc NP s=2 e=4> <!-- the man -->
<arc PP s=4 e=7> <!-- with the telescope -->
<arc P s=4 e=5> <!-- with -->
<arc NP s=5 e=7> <!-- the telescope -->
</parse>
</sentence>
```

Note that SGML's optional SHORTTAG feature was used in each "arc" start-tag above to omit the attribute name "marker" (and the separating equals sign) preceding the value of this attribute. This form of markup minimization can be used when the attribute's declared value is a name token group—an explicit list of possible values.

The above encoding does not express the relationship between words in the original sentence and the endpoints of the arcs. ID and IDREF attributes can be added to the general method to add this information. The replacement declarations are shown below:

```
<!ELEMENT text (word)+ >
<!ELEMENT word (#PCDATA) >
<!ATTLIST sentence id ID #IMPLIED
<!ATTLIST word id ID #IMPLIED
<!ATTLIST arc marker (s, np, vp, pp, v, n, p) #IMPLIED
          s IDREF #REQUIRED
          e IDREF #REQUIRED >
```

Here, the "id" attribute for "sentence" represents the point before the first word and the "id" of each word identifies the point following it. The words themselves are good candidates for the ID values. Since the word "the" occurs twice in the sample sentence, different values must be chosen for each. The revised markup is:

```
<sentence id="s1">
  <text>
    <word id=I>I</word>
    <word id=saw>saw</word>
    <word id=the1>the</word>
    <word id=man>man</word>
    <word id=with>with</word>
    <word id=the2>the</word
    <word id=telescope>telescope</word>
  </text>
  <parse>
    <arc S s=s1 e=telescope>
    <arc NP s=s1 e=1>
    <arc VP s=I e=telescope>
    <arc VP s=I e=man>
    <arc V s=I e=saw>
    <arc NP s=saw e=man>
    <arc PP s=man e=telescope>
    <arc P s=man e=with>
    <arc NP s=with e=telescope>
  </parse>
  <parse>
    <arc S s=s1 e=telescope>
    <arc NP s=s1 e=I>
    <arc VP s=I e=telescope>
    <arc V s=I e=saw>
    <arc NP s=saw e=telescope>
    <arc NP s=saw e=man>
    <arc PP s=man e=telescope>
    <arc P s=man e=with>
    <arc NP s=with e=telescope>
  </parse>
```

```
</sentence>
```

Since the ID values have been chosen to indicate the marked phrase, the comments from the previous version have been omitted.

This solution is similar to the linguistic analysis tags described in the Guidelines. As is the case in the example, the "fStruct" element allows multiple analyses of the same content.

4.4 Application-Specific Content

Of course, any special notation expressed as a character string can be entered into an SGML document, although the SGML parser will not be able to validate the notation. The ambiguous sentence could be coded as follows:

```
<sentence>
  <text>I saw the man with the telescope.</text>
  <parse p='( (1) ( ((2)(3 4)) ((5)(6 7)) ) )'>
  <parse p='( (1) ( (2) ( (3 4) ((5)(6 7)) ) ) )'>
</sentence>
```

Or:

```
<sentence>
  <text>I saw the man with the telescope.</text>
  <parse p='s( np(1) vp( vp(v(2) np(3 4)) pp(p(5)np(6 7)) ) )'>
  <parse p='s( np(1) vp( v(2) np( np(3 4) pp(p(5)np(6 7)) ) ) )'>
</sentence>
```

4.5 Ambiguities as Non-hierarchical Material

The final suggestion for encoding ambiguities is simply to treat the alternatives as material that supplements the document's primary hierarchical structure, using one of the methods discussed in Section 2 above.

5 Overlapping Structures

Sometimes the boundaries of hierarchical units are not clear. Examples include:

- "She took advantage of Joan" (where "of" can be part of the single unit "took advantage of", or part of the prepositional phrase "of Joan")
- "Broadway Hit or Miss"
- Apo koinu constructions

Any of the methods discussed in Section 1 of this paper can be used to encode both relationships.

6 Synchronizing Parallel Structures

The next issue is encoding parallel texts in order to show corresponding segments. Examples include:

- translations, for example, into the nine languages of the EEC
- manuscript variants (for example, of the Bible) or recensions
- phonemic and orthographic transcriptions of the same content

6.1 Implicit Parallelism

As long as all versions of the text present corresponding constructs in the same order, all variations of a segment can be encoded as the subelements of a single unit. The DTD must be defined to show the alternative structures at the lowest level of the hierarchy. For example, the following declarations can be included in a DTD to indicate the corresponding phonemic and orthographic transcriptions of the words in sentences:

```
<!ELEMENT segment - O (phonemic, orthographic) >
<!ELEMENT phonemic - O (#PCDATA) >
<!ELEMENT orthographic - O (#PCDATA) >
```

The encoding of a particular sentence might begin like this:

<sentence>
  <segment>
    <phonemic> (phonemic transcription of 'the')
    <orthographic>the
  <segment>
    <phonemic> (phonemic transcription of 'fat')
    <orthographic>fat
  <segment>
    <phonemic> (phonemic transcription of 'cat')
    <orthographic>cat
...

Note that end-tags for "segment", "phonemic", and "orthographic" are omitted above. SGML allows this form of markup minimization when a parser, following a strictly defined algorithm, can detect the end of the element without it, and the optional OMITTAG feature has been selected.

6.2 Synchronizing with Unique Identifiers

When the order of segments varies from one text to another, the correspondence can be indicated with ID and IDREF attributes. In the following translation, successive phrases from the first language are numbered so they can be referenced in the other version:

```
<sentence>
  <translation German>
    <seg id=p1>Tor
    <seg id=p2>nach Durchfahrt
    <seg id=p3>bitte
    <seg id=p4>zumachen!
  </translation
  <translation English>
    <seg id=p3>Please
    <seg id=p4>close
    <seg id=p1>the gate
    <seg id=p2>after passing through!
  </translation>
</sentence>
```

A more elaborate version of this same mechanism, allowing for one-to-many matching of segments, is found in the alignment mechanism of Chapter 14 of the Guidelines.

6.3 Synchronizing with Analysis Elements

The segments of parallel texts may appear in different orders without a one-to-one matching in the different versions. In this case, the "alignment" element illustrated in Section 2.4 can be used.

7 Cross-references

The next problem is how to refer to locations elsewhere in the same document or in a separate document. ID and IDREF attributes provide a natural method of coding internal cross-references. For example, if a DTD contains the following declaration:

```
<!ATTLIST setting
  id ID #IMPLIED
  same IDREF #IMPLIED
>
```

a document instance might contain markup such as:

```
<act>
<scene><setting>A heath</setting>...
<scene><setting>A camp near Forres</setting>...
<scene><setting>A heath near Forres</setting>...
<scene><setting id=palace>Forres. The palace</setting>...
<scene><setting>Inverness. Macbeth's castle</setting>...
<scene><setting>Before Macbeth's castle</setting>...
<scene><setting id=within>Within Macbeth's castle</setting>...
```

```
<act>
<scene><setting same=within></setting>...
<scene><setting same=within>The same</setting>...
<scene><setting same=within>The same</setting>...
<scene><setting>Outside Macbeth's castle</setting>...
<act>
<scene><setting same=palace>...
```

As with all other ID and IDREF attributes, the SGML parser verifies that the value of attribute "same" is always the value of an ID attribute somewhere in the document instance. Aside from checking for errors, however, it does no special processing of the attribute. If, for example, the setting of Act II, Scene I is to be printed as "Within Macbeth's castle" by copying the text ¿from the cross-referenced setting, it is the responsibility of application software to copy this text.

Since the value of an IDREF attribute must be the value of an ID attribute elsewhere in the same document instance, IDREF attributes cannot be used for external cross-references. However, an attribute with the declared value NAME can be used to enter an ID value from another document. The external document can be identified by a second attribute of the same element. The declared value of this attribute can be ENTITY, to indicate that the external document is identified by the external identifier in an entity declaration, or it can be NAME or CDATA, to indicate that the external document is identified by a string that the application knows how to interpret. The use of the ENTITY attribute is illustrated below:

```
<!ENTITY macbeth SYSTEM "macbeth.doc" SUBDOC>
<!ENTITY hamlet SYSTEM "hamlet.doc" SUBDOC>
...
<setting play=hamlet id=graveyard> and <setting play=macbeth
id=castle> have much in common
...
```

When the possible external documents are known in advance, a variation of this scheme can build the document identifiers into generic identifiers in the DTD. The above example could then be coded as:

```
<hamlet id=graveyard> and <macbeth id=castle>
have much in common...
```

8 Vagueness of Location

The final issue addressed in this paper is encoding segments with endpoints that are not well defined. Researchers may wish to encode details such as an echo of another text, which may begin and end gradually, so that the markup indicates a section that is certainly an echo, surrounded by a penumbra that may or may not represent an echo. Thus, the researcher tagging the document must be able to express opinions such as:

- "the passage begins somewhere along here"

- "the passage begins somewhere between these points"

8.1 Precision Attributes

14

Add a "precision" attribute to the element whose location is uncertain. The possible values of this attribute can be keywords such as "vague", "definite", "probable" or can be integers in which greater values indicate the tagger's greater confidence.

8.2 Marking Text Ranges

Mark both the start and end of the range of text where each endpoint of the passage may occur. The range can be marked by elements with declared content of EMPTY:

```
<echostart range=start>...<echostart range=end>
...
<echoend range=start>...<echoend range=end>
```

Alternatively, nested elements can be used:

```
<penumbra>
...
<echo>...</echo>
...
</penumbra>
```

or

```
<echo status=possible>
...
<echo status=certain>...</echo>
...
</echo>
```

9 conclusions

The TEI metalanguage committee explored several practical problems of text encoding and illustrated SGML solutions that are compatible with TEI encoding. This paper has presented several alternative techniques applicable to a range of issues and to classes of documents ranging from particular editions of historical works, through grammatical analyses, to transcripts of interviews and meeting minutes. Instead of recommending a particular approach, the paper has provided multiple alternatives, giving the reader criteria that can be used to select the variation must suited to requirements of particular projects. In the process of describing these techniques, the paper has introduced some constructs of SGML

[14] See also Chapter 17 of the Guidelines.

not covered in other TEI papers. The examples illustrate that SGML's flexibility allows it to capture the essential hierarchical nature of text while at the same time addressing properties that do not fit neatly into the hierarchy.

The ideas in this paper influenced the work of committees and work groups[15], and can serve as useful background for researchers intending to extend the Guidelines for marking other non-hierarchic structures.

[15] And of the two editors, who were involved in these discussions.