Lessons Learned from Using SGML in the Text Encoding Initiative
Technical Report 95-375

David T. Barnard*
Department of Computing and Information Science
Queen's University
Kingston, Ontario K7L 3N6
Canada
David.Barnard@QueensU.CA

——

Lou Burnard
Oxford University Computing Services

——

C. Michael Sperberg-McQueen
The University of Illinois at Chicago

March 2, 1995

Abstract

In April of 1994 the ACH-ALLC-ACL Text Encoding Initiative published *Guidelines for Electronic Text Encoding and Interchange* (Document TEI P3). SGML was used as the basis for the encoding scheme that was developed. Several innovative approaches to the use of SGML were devised during the course of the project. Three aspects of this innovation are documented in the paper. First, all of the tags are organized into sets that can be included easily into the project DTD, which allows the corresponding features to be used in documents only when required. Second, mechanisms were developed to relate parts of documents in non-hierarchic ways. Third, a mechanism was developed to allow extension of the DTD in a disciplined manner. We comment on the effectiveness with which SGML could be used in these ways and the shortcomings we perceive.

*This paper was prepared while the first author was on sabbatical at INRIA (Institut National de Recherche en Informatique et en Automatique), Rocquencourt, France.

Contents

1 Introduction

In April of 1994 the Text Encoding Initiative published *Guidelines for Electronic Text Encoding and Interchange* (Document TEI P3)[1]. The 1290 pages of this document are the result of several years of collaborative effort by members of the research and academic community in Europe, North America and Asia to derive a common encoding scheme for complex textual structures.

The project grew out of an organizational meeting held in 1987 at Vassar College in Poughkeepsie, New York. Participants at that meeting were primarily researchers involved in the collection, organization, and analysis of electronic texts for linguistic and humanistic research. There was a shared view that the enormous variety of mutually incomprehensible encoding schemes in use was a hindrance to research. A new scheme was required that would allow the interchange and reuse of texts, and would give guidance for those creating new electronic texts.

A project was created with the joint sponsorship of the Association for Computers and the Humanities, the Association for Computational Linguistics, and the Association for Literary and Linguistic Computing. Funding was obtained from the National Endowment for the Humanities (USA), Directorate XIII of the Commission of the European Communities, the Social Science and Humanities Research Council of Canada, the Andrew W. Mellon Foundation, and the various organizations—companies, government agencies, universities and institutes—where the participants worked.

Participants in the project were organized into committees and work groups focussed on specialized topics, each of which contributed material for the Guidelines. A Steering Committee, a Technical Committee, and the two editors (Burnard and Sperberg-McQueen) provided overall coordination.

---

[1] *Guidelines For Electronic Text Encoding and Interchange (TEI P3)*; C.M. Sperberg-McQueen and Lou Burnard, eds.; ACH-ACL-ALLC Text Encoding Initiative, Chicago and Oxford, May 1994.

For those readers with access to the World Wide Web, the Guidelines can be found online in several places. The official project repository is at

ftp://ftp-tei.uic.edu/pub/tei

(for users in North America) and its mirror sites

ftp://ftp.ifi.uio.no/pub/SGML/TEI

(for users in Europe) and

ftp://TEI.IPC.Chiba-u.ac.jp/TEI/P3

(for users in Asia), or at

ftp://info.ex.ac.uk/pub/SGML/tei

or at

http://etext.virginia.edu/TEI.html

in a form that can be searched.

This section is heavily based on the relevant material in the *Guidelines*.

The new encoding scheme was required to:

- adequately represent all the textual features needed for research,
- be simple, clear and concrete,
- be easy for researchers to use without special-purpose software,
- allow the rigorous definition and efficient processing of texts,
- provide for user-defined extensions, and
- conform to existing and emerging standards.

SGML was used as the basis for the encoding scheme that was developed. The Guidelines can thus be viewed as a complex application of SGML, in effect a large DTD design. Figure 1 shows a simple document encoded using the TEI DTD. SGML was a new development when the project began, and was certainly not a technical artefact with which the members of the target community were then familiar. Accordingly, to promote acceptability of the scheme, it was necessary to ensure that:

- a common core of textual features could be easily shared,
- additional specialist features could be easily added,
- multiple parallel encodings of the same feature were possible,
- users could decide how rich the markup in a document would be, with a very small minimal requirement, and
- there should be adequate documentation of the text and the markup of any electronic text.

To achieve these desiderata, several innovative approaches to the use of SGML were devised during the course of the project. Three aspects of this innovation are documented in the paper. In subsequent sections we discuss techniques used to support modularization of the DTD, techniques for encoding non-hierarchical structural information, and techniques for disciplined extension of the project DTD. We conclude the paper with some recommendations for future developments of SGML.

2 Towards A Modular DTD

2.1 Behaviour

Since the interests of the community represented by the project are very broad, the documents to be encoded according to the TEI Guidelines are drawn from a wide range

```
<!DOCTYPE tei.2 system 'tei2.dtd' [
  <!ENTITY % TEI.prose 'INCLUDE'>
  <!ENTITY english.wsd system 'teien.wsd' SUBDOC>
]>
<tei.2>
<teiHeader>
  <fileDesc>
    <titleStmt><title>Short document.</title>
    <publicationStmt><p>Unpublished.</p></publicationStmt>
    <sourceDesc><p>Electronic form is original.</p></sourceDesc>
  </fileDesc>
  <profileDesc>
    <langUsage><language id=eng wsd=english.wsd usage='100'></langUsage>
  </profileDesc>
</teiHeader>
<text>
  <body>
    <p>A very short TEI document.</p>
  </body>
</text>
</tei.2>
```

Figure 1: A Simple TEI Document

- provision for characters sets to be used in documents
- tags for specifying the TEI header
  (including the file description, encoding description)
- tags for specifying basic elements
  (including paragraphs, emphasis, quotations, list notes, simple pointers, references)
- simple text structure
  (including front and back matter, possibly nested divisions)

Figure 2: Core Features

of fields. This has resulted in a large and complex set of specifications for tags and their inter-relationships. No single document, though, is likely to need all of the tags that are defined in the DTD.

Accordingly, the features to be encoded are arranged in several categories, and the DTD is structured to allow features to be included in documents or excluded from them.

There is a *core* set of features that appear in all documents. See Figure 2 for the features in the core. These core features are represented in two tag sets. One of these contains the tags for specifying the TEI header, which is a description of the electronic text required in every TEI document (see Figure 1 for a simple example). The other is a tag set for representing basic elements like paragraphs. Simple text structures are represented by elements which may differ in each base, but a default set of elements is provided for this purpose as part of the core.

This is, in fact, the defining aspect of the "core"—it is present in all of the bases.

The base tag sets correspond to the major document categories. See Figure 3 for a list of these, together with some examples of what each contains. Each TEI document is drawn from one of these categories[2] and thus each document instance must select one of these tag sets. Each of these bases includes the core features either by explicitly including the core tag sets or by encapsulating the core features in the expression of the tags in the base set itself.

Tag sets for additional features can be added to the base as required, depending on the information to be represented in the text, and the use to which the text will be put. See Figure 4 for a list of these, together with some examples of what is included in each.

As a concrete example, to use the TEI DTD to encode a manuscript of a play, one would choose the drama base (which implicitly includes the core features). If at a later time the same or another researcher wished to add some references from this play to other documents,

[2]In addition two special purpose bases are provided which enable elements from different bas sets to be combined int he same document, either promiscuously, or in a controlled way (see section 3.4 of the *Guidelines*).

- prose
  (contains only the core features)
- verse
  (structure within lines, rhyme, metrical structure)
- drama
  (cast lists, performances, stage directions)
- transcriptions of speech
  (utterances, pauses, temporal information)
- print dictionaries
  (structure of entries, grammatical and typographical information)
- terminological databases
  (terms and definitions)

Figure 3: Base Tag Sets

and to add some information about textual sources for the play, the tag sets for linking and for text criticism could also be used (see Figure 5).

2.2 Mechanism

There is no single feature in SGML to support this kind of modularization directly. Instead, the TEI DTD is constructed using a large number of marked sections, one for each tag set. Each of these marked sections is guarded by a parameter entity. The default value of these parameter entities is IGNORE. The user must decide explicitly which parameter entities to set to INCLUDE in the declaration subset. Figure 6 shows an example of this, with the drama base and the additional tag sets for simple analysis and for segmentation and linking being chosen.

2.3 Evaluation

Using SGML in this manner does achieve the kind of modularization that the TEI was seeking. It is possible to have different bases and additional tag sets that are added orthogonally.

However, there are a number of things to be said against it.

- The DTD must name all tag sets explicitly, whether or not they are used in a particular document. This makes extending the DTD with new tag sets difficult in that any such extension will invalidate existing copies of the DTD.

- linking, segmentation and alignment
  (extended pointers, segments, anchors)
- simple analytic mechanisms
  (linguistic segments, spans of text)
- feature structures
  (designed for linguistic analysis but useful for recording any values of any named features detected in the text)
- certainty and responsibility
  (who tagged this and how certain is the encoding?)
- transcription of primary sources
  (deletions, illegibility)
- critical apparatus
  (sources for the text, attaching witnesses to spans of text)
- names and dates
- graphs, networks and trees
  (encoding data structures to represet information about text)
- tables, formulae and graphics
- language corpora
  (collections of text, sources)

Figure 4: Additional Tag Sets

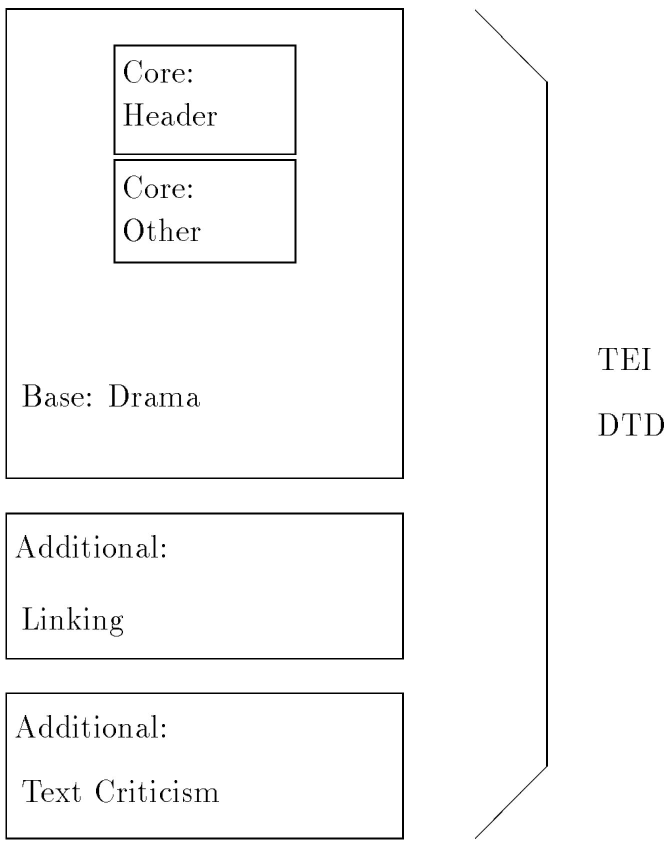

Figure 5: Combining Tag Sets

```
<!DOCTYPE TEI.2 system 'tei2.dtd' [
  <!--  base tag set             -->
  <!ENTITY % TEI.drama    'INCLUDE'>
  <!--  additional tag sets      -->
  <!ENTITY % TEI.textcrit 'INCLUDE'>
  <!ENTITY % TEI.linking  'INCLUDE'>
]>
```

Figure 6: Combining Tag Sets in the Declaration Subset

- There is no scoping of names in the different tag sets, so names must be unique across all tag sets. For a large project this may be difficult to ensure if tag sets are being defined by different people at different times; the TEI overcame this problem by devolving responsibility for technical coherence of the scheme to the two editors of the project.

- There is no indication in an encoded document of how the tags are grouped into tag sets. The tags all appear to be drawn from one homogeneous name space. In fact, they *are* from one name space so far as any SGML processor is concerned.

In short, this kind of modularization is possible in SGML, but it is not explicitly supported by language features or mechanisms. The idea of managing parts of large DTDs in a modular fashion does not appear to have been in mind during the design of SGML.

3 Handling Non-Hierarchic Structures

3.1 Behaviour

Within the applications envisioned for the TEI DTD, there are many cases in which parts of a document need to be related in complex ways.

- One part of a document makes reference to another part of the document. In simple cases this can be handled by the SGML mechanism for assigning identifiers to elements, and making reference to these identifiers elsewhere in the document.

- One document makes reference to an element in another document. There is no mechanism in SGML for doing this, although HyTime does support it.

- The target of a pointer might be a point or a span of text, not necessarily co-extensive with some element or elements.

- Parallel texts need to be aligned. For example, an English text with a French translation could be aligned so that each paragraph of one is linked to the appropriate paragraph of the other. It is not necessary, of course, to be restricted to two sets of parallel material.

- In the case just described the texts might be expected to have parallel structures. However, other kinds of correspondences can occur as well. For example, if one wants to represent two or more different possible syntactic analyses of a natural language sentence, the text will be the same but the structures will be different.

- In the transcription of speech or the presentation of drama it can be necessary to indicate that two texts were, or are to be, spoken at the same time, so that a temporal correspondence must be achieved.

- An element might be aggregated from various pieces of a document occurring elsewhere. For example, a piece of quoted text might be interspersed with comments, but the entire quoted segment is to be indicated in some aggregation element.

- An entire document might be represented with more than one structure. In essence, this means that more than one parse tree spans the same string (has the same frontier). This can be the case if a verse drama is to be encoded so as to represent both the speakers and their speeches, and the verse lines and stanzas. Neither of these hierarchies contains the other. The SGML CONCUR feature appears to provide for this, but it is verbose, requires modification of existing tags if new structures are added to the encoding (to specify the DTDs to which an element pertains), and is not implemented by most SGML software systems[3].

3.2 Mechanism

The TEI uses a number of different encodings to overcome the problems addressed above. The basis for these is the use of the SGML ID and IDREF mechanism to link and align elements in various ways and with different structures. The ID mechanism is usually used in conjunction with other attributes added to tags to specific semantics that indicate the nature of the links being built.

However, not all the places to which pointers are to be attached can be assumed to have SGML IDs. A system of specifying pointers was devised for the TEI for these cases. These pointers are called *extended pointers* because they are extensions of the SGML ID mechanism. These can be used to point within a document or to another document.

While this system not surprisingly is similar in some ways to HyTime (indeed, TEI members were active in the HyTime effort and the two projects influenced each other), it is simpler and easier to implement. The extended pointer mechanism incorporates ideas from :

- the SGML ID mechanism ;
- HyTime ;
- HyQ, the HyTime query language (so that pointers may need to be dynamically evaluated) ;
- references up and down in a hierarchy (as in a hierarchical file system, but relative to the parse tree of the document) ;
- other structural relationships (such as finding the nth sibling after the node where the pointer exists) ; and

[3]See Makoto Murata ; File Format for Documents Containing Both Logical Structures and Layout Structures ; *Electronic Publishing* 1 (1995) (to appear).

For an older discussion of some of the problems, see D. Barnard, R. Hayter, M. Karababa, G. Logan and J. McFadden; SGML-Based Markup for Literary Texts: Two Problems and Some Solutions; *Computers and the Humanities* 22 (1988) 265-276.

- pattern matching of a relatively simple sort.

The pointer mechanism can be used to solve various linking and alignment problems in texts. Some of the uses described above are straightforward applications of the pointer mechanism, while others are complex enough to warrant their own structure and tag set. For many of these, a *link* and *link group* are used.

3.3 Evaluation

The ID mechanism of SGML is sufficient for simple references to named elements in a single document. Pointing outside the document, or specifying pointers in other ways (using structural relationships, patters, or queries) is not possible in SGML without some agreed conventions. HyTime is one way to extend the SGML mechanisms.

The TEI devised its own extensions, comprising a set of conventions for interpreting some tags and some attributes of tags.

The mechanism used in the TEI is sufficient for encoding the various structures for which it was intended.

However, there are a number of problems with the TEI solution.

- It goes beyond SGML, and relies on a set of conventions about the interpretation of attributes rather than any formal mechanisms in SGML[4].
- There are too many different ways of encoding non-hierarchical information in the TEI. With some effort—probably more a political effort of having various users agree than a technical effort to demonstrate feasibility—it might be possible to simplify the TEI DTD to have fewer structure-building tags.
- Some simple notions are not easily stated. For example, the set of bidirectional links needed to specify the alignment of a text with its translation is a relatively simple thing conceptually, but its specification requires a good deal of markup. We hope that this markup might eventually be generated programmatically, but we do not know of existing tools that could generate it for TEI documents.

Modern documents have many non-hierarchical features, and SGML is very strongly oriented toward encoding hierarchy. In fact, it easily encodes only a single hierarchy.

[4]This is also true of HyTime, of course, although it has the advantage of being an ISO standard and the disadvantage of being much more complex than the TEI solutions.

4 Modifying the DTD

4.1 Behaviour

Although the Guidelines support the encoding of many different kinds of documents, it is expected that users in the community will need to extend the DTD with definitions appropriate for specialized projects. The modifications needed to be done in a documented manner, and in such a way that they are able to be checked by a parser using SGML mechanisms.

The particular modifications that are supported are these:

- deleting an element: a user may want to prohibit the use of a tag in an included tag set (or, indeed, in the core) which it should be possible to do by telling the SGML parser that the tag is not available;
- renaming an element: a user may want to use a different name for a tag because of local conventions, or perhaps may want to systematically rename all of the tags into another language than English;
- extending a given class: some of the tags fit together into classes that may, for example, share common attributes, and it should be possible to extend one of these classes by adding more tags to it; and
- giving a new content model: a user may want to redefine the structure of an existing tag, perhaps by putting a structure into something that is only specified as character data in the TEI DTD, or to add a new tag and indicate how it is to be included in the existing hierarchy.

4.2 Mechanism

SGML does not provide mechanisms explicitly for these purposes. Instead, these modifications are enabled by a use of parameter entities in the DTD.

There are two versions of the TEI DTD (actually, of the TEI tag sets). The first is a *publication version* which is intended to be human readable. It is the version that is used in the discussion in the Guidelines. There is a second *modifiable version* that is programmatically derived from the first.

The second version differs from the first in that parameter entities have been introduced to define various aspects of the expression of the DTD itself. These parameter entities can be redefined in the declaration subset for a document, and through careful use of this redefining ability, the modifications accomplished.

For example, in the modifiable version there is a guarded section for each element definition. A parameter entity is used as the guard on the marked section. The default value of the parameter entity is INCLUDE. Specifically, for the *p* tag (for paragraphs) the modifiable version would have

```
<![ %p [
  <!-- element and attlist declarations for p go here -->
]]>
```

If a user wants to delete this tag, the value of the parameter entity—with the same name as the tag—is set to IGNORE, like this

```
<!ENTITY % p IGNORE>
```

Since SGML allows parameter entities to have several definitions, and uses the first definition encountered, if this redefinition occurs in the declaration subset it overrides the default given in the external DTD file.

As a second example of how these parameter entities work, consider renaming the *p* tag. The modifiable version of the DTD includes a parameter entity that contains the name of the tag. If a user wants to rename the tag, that parameter entity—it has the name n.foo where foo is the name of the tag itself–is redefined, like this

```
<!ENTITY % n.p 'paragraph'>
```

To give a new content model for an existing tag, one first deletes the existing element definition using the mechanism described above, and then gives the new definition in the declaration subset. Of course, in the extreme, a user could delete all of the existing TEI definitions and essentially create a new DTD. This kind of irresponsible anti-social behaviour is deprecated.

There are other things that could be supported by an even more complex set of parameter entities, like changing of attribute names, adding of an attribute to an attribute list, redefining an attribute, changing the inclusion and exclusion exceptions applied to a definition, and so on. While the details of how these things could be done were worked out, the mechanisms were not included in the TEI DTD because of their complexity and the rarity of their expected use.

4.3 Evaluation

A mechanism has been devised that allows required modifications to the TEI DTD to be done easily and in a controlled manner, so that an acceptable range of changes is defined and all acceptable changes are communicated within the SGML formalism and can be checked by a parser.

However, there are a number of things to be said against the approach.

- Although an SGML parser can be used to validate the changes, non-SGML software had to be written to transform the publication version of the DTD into the modifiable version. We step outside the SGML formal system at that point.
- The modifiable DTD is difficult to read.
- The mechanism is difficult to explain to many of the potential users. We simply ask them to take it on faith. If the mechanism were completely hidden from them, this would be fine, but if they read the DTD or make errors and get diagnostics from software systems, these can be almost impossible to interpret.
- Renaming tags in another language is possible but not entirely satisfactory, in that attributes and attribute values cannot be easily renamed.
- The redefinition mechanism is unconstrained. Once the door is opened up to redefinition in this way, there is no way to place a limit on it, as was mentioned earlier. Deprecation is only social pressure not technical prohibition. We would like to have a tighter control on what it means to be TEI conformant.

5 Conclusion

The DTD developed by the Text Encoding Initiative is large and complex. It is inherently modular, and is are known to be insufficient for all the purposes for which it will be used, so that extension is necessary. The documents to be described with this DTD are also inherently complex in several ways, including their use of non-hierarchical structures that need to be encoded.

The TEI was thus in several ways pushing the limits of what has been done with SGML. While mechanisms were devised that meet the technical requirements, these mechanisms are less than satisfactory in several respects in that they are complicated and are not directly supported as SGML features. SGML is contorted and stretched for these purposes.

The extended pointers devised by the TEI can be thought of as simplified versions of what HyTime provides. Positively, we can say that HyTime is filling a real need. Negatively, we might have hoped that SGML itself would provide a more powerful pointer mechanism so that users would not be driven to HyTime for relatively simple applications.

All of the problems with DTD modularization and extension that arose in the Text Encoding Initiative could be easily and straightforwardly handled if SGML provided some kind of abstraction mechanism for DTDs. If DTDs were objects that could be extended with names that could be overloaded, all that the TEI required could have been easily accomplished. Adding such a feature to SGML does not appear to us to be difficult.

Acknowledgement

The Text Encoding Initiative, including the aspects of it reported here, is the result of the collaborative work of many scholars and technical experts. The introductory material in the *Guidelines* names all of these persons.