Specifying Document Structure: Differences in LaTeX and TEI Markup

C. M. Sperberg-McQueen
ACH/ACL/ALLC Text Encoding Initiative
University of Illinois at Chicago
Internet: U35395@uicvm.cc.uic.edu
Bitnet: U35395@UICVM

Abstract

LaTeX and the Standard Generalized Markup Language (SGML), specifically the SGML tag set created by the Text Encoding Initiative (TEI), are two major systems developed to make it easier to create and verify valid documents. Each attempts to specify and enforce explicit definitions of valid textual structures; each faces questions regarding the structural components of texts, as well as the choice of abstract structures for representing and of formal notations for specifying them.

This paper focuses on the ways LaTeX and the TEI identify and classify the structural and other components of text; discusses the models of text underlying the two systems and the methods of text definition and validation they make possible; describes a number of specific issues that arise; considers some systematic differences; and describes one possible way in which they might coexist.

Introduction

As mechanical processing of text becomes easier, it also becomes easier — and more important — to specify formally what a text is and to use that specification to ensure the validity of the data stream that represents the text in the machine. Validation becomes important because application software uses increasingly complex data structures for text representation, and because our mechanical processing can destroy or corrupt data with an efficiency and thoroughness that far exceed the wildest dreams of the most assiduous scholar working by hand. Validation has become easier because computer science has provided a rich set of data structures to use in representing texts and increasingly sophisticated notations for specifying the valid forms of those data structures.

Today I want to discuss the specification of document structure in LaTeX and in the SGML tag set defined by the ACH/ACL/ALLC Text Encoding Initiative (TEI), an international effort to define an application-independent, language-independent, system-independent markup language for general use (especially in research). This has four parts: First, I'll discuss the substantive questions of what the structural components of texts are; and, second, the methodological questions of choosing abstract structures with which to represent texts and formal notations with which to specify the abstract structures. Third, I'll describe briefly a number of concrete problems in the proper application of such abstract structures and formal notations to pre-existing texts of the sort studied by most textual scholars, and, finally, I'll describe how I think SGML and LaTeX can usefully coexist in practice.

Any text-encoding scheme must provide ways to represent the characters of a text, its basic structure, intrinsic features other than structure, and extrinsic information associated with the text by an annotator. I am here concerned not with the first of these, but only with the other three.

Substantive Issues: What Belongs in a Text?

Basic text structure. On the basic structural components of text, there is a rather surprising agreement among the various markup languages in

current use — at least among those which attempt to assign structure to texts.

LaTeX implicitly divides a text into a title page (created by the `\maketitle` command, which must be preceded by author, document title, and similar information), followed by the text body and, optionally, by back matter (marked with the `\appendix` command). The body and back matter comprise either undivided text or a series of `\parts` or `\chapters`. Within parts, there is a straightforward hierarchy of chapter, section, subsection, subsubsection, paragraph, and subparagraph, in which the hierarchical relationships are enforced automatically.

The TEI tag set similarly divides documents into front matter (which can contain more than the title page), body, and back matter, with body and the parts of the front and back matter all divided into hierarchically nested blocks of text. Since existing (historical) texts may use structural units with names other than *chapter*, etc., TEI uses the generic term *div* for these blocks of text: The text body is a series of `<div0>`s, divided into `<div1>`s, divided into `<div2>`s, etc. The user can specify what name should be associated with a given level by giving the name as the value of an SGML attribute on the tag; for example, `<div1 name='Chapter'>`. The current draft stops at `<div5>`, but this is a purely arbitrary decision and can be changed.

An alternative proposal (used in some existing SGML tag sets) is to eliminate the redundant nesting-level numbers and replace `<div0>` through `<divN>` by the single tag `<div>` or `<block>`. Since the nesting level can be readily calculated at processing time, blocks at different levels can be processed differently. This is elegant but complicates life for whoever is specifying the processing.

Lower-level floating structures. Within the main structural divisions of the document, text is divided into paragraphs, and these have no visible internal formal structure. There are some chunks of text, however, that do have visible internal structure; these I call *crystals*, borrowing a term from Steven J. DeRose (in a TEI working paper). Crystals are internally structured free-floating units of text, such as figures, tables, or bibliographic citations. Leslie Lamport calls (some of) them *floating bodies*.

LaTeX and the TEI recognize roughly the same set of large-scale crystals: lists, verbatim examples, displayed equations, figures, tables, and bibliographic references. The TEI further expects to provide tags for marking much smaller crystal structures like dates, addresses, personal and corporate names, and so on. This reflects a major difference between LaTeX and the TEI: LaTeX does not need special markup for addresses or personal names, because these do not typically require special treatment in document layout. The closest LaTeX gets are with the conventions used by BibTeX to distinguish first names from last names based on where one puts the comma. The TEI is not exclusively or primarily concerned with producing hard copy from documents, but with making it possible to mark the documents' logical structure in support of *whatever* kind of processing the user might want to do. Historians, librarians, office-automation people, and others may all want special processing based on the internal structure of names and dates — not for printing, perhaps, but for indexing or other reasons.

For the converse reason, the TEI has not yet made any concerted attempt to provide yet another language for the description of mathematical equations, figures, graphics, or tables. LaTeX, being concerned with processing for output (as well as with the logical structure of the text), can hardly get by without providing markup for such crystals. The TEI has thus far exploited a feature of SGML that allows sections of the text to be marked up in non-SGML notations so they can be processed by some appropriate processor. This keeps SGML out of the graphics-standards wars and allows designers of SGML tag sets to stay out, too. Although tables often have a clear logical structure, and it would make sense to attempt to capture this in descriptive markup, the TEI has yet to make any concrete recommendations in this area; this is an area of ongoing work.

For bibliographic citations, the TEI provides a structured form patterned on the standard forms for bibliographic references developed by librarians, as well as a much less tightly structured form for those with less concern about database usage of their citations. The structured form provides more structure than appears to be available in the prose segments of LaTeX documents, but is less rich than the corresponding BibTeX structure. This is an area in which the TEI tags must definitely be extended to at least the level of detail offered by BibTeX.

Phrase-level attributes. Within the paragraph, the rigid hierarchical text structure of chapter, section, subsection, etc., suddenly breaks down, and we are confronted with a non-rigid mess with the consistency of soup. Within this soup, some larger chunks (crystals, like figures and tables) may be

floating that we've already discussed. Some nonstructured bits may be floating there as well: emphasized phrases, quotations, and the like. Here, LaTeX and the TEI take a very similar approach. Instead of describing the visual presentation of the text in a particular output medium, both encourage the user to describe its logical characteristics. Thus, LaTeX provides an `\em` command for emphasized text and suggests that the `\bf`, `\sc`, and similar commands "should appear not in the text but in the definitions of the commands that describe the logical structure." Similarly, the TEI provides several tags for marking words, phrases, or passages that are specially marked in some way:

- emph
- foreign
- cited word
- term
- book or journal title
- quotation
- 'scare quotes'
- article or poem title

In addition, since for historical texts one doesn't always know why something is presented in a different font, one can also mark such material simply as `<highlighted>` without any attempt to explain why. This is a necessary compromise between the advantages of logical or descriptive markup and the requirements of scholarly integrity.

Typographic details, layout, processing. Treatment of typographic details, layout, and similar matters is predictably far more elaborate in LaTeX than in the TEI tag set. LaTeX, even with its explicit preference for *logical* document design over *visual* design, does after all have the function of providing good typeset output; since good typesetting is not wholly algorithmic, TeX and LaTeX provide plenty of opportunities for the user to give them hints on what the output should look like.

The TEI tag set is far poorer in this respect, for two reasons: First, we are attempting to create an application-independent markup scheme, suitable for many different types of processing. It seems more important just now to stress the possibilities for processing other than printing, because these are so often overlooked. Trying to provide a rich set of layout tags in the first draft would invite serious misunderstandings and suggest that the TEI was trying to compete with TeX and other typesetting systems. The second reason is that SGML is designed as a declarative, not a procedural language — precisely to ensure the application independence it is designed to achieve. It *is* possible to specify presentation declaratively rather than procedurally, as we do already with the `<highlighted>` tag described above. But a full declarative description of page layout is a large, challenging assignment, one that requires a lot of further work. It is also a task that the International Organization for Standardization (ISO) is already addressing with its Document Style and Semantics Specification Language (DSSSL); if the DSSSL project is successful, the TEI can piggyback on their success by basing its further work on layout problems on DSSSL.

Annotation. LaTeX provides useful tools for annotation: footnotes, marginal notes, and (in SliTeX) inline display notes. These correspond directly to a single TEI tag, `<note>`, that uses an attribute value to specify its location or type. But the TEI provides a large number of other tags for annotation of various kinds that do not appear in LaTeX:

- an extensive document header that documents the electronic text: its date and place of origin, names of those responsible, copy text used, and specifics of the encoding used;
- tags for special items, like dates and numbers, that allow their values to be given in a standard format (so that a note containing the sentence, "Let's have lunch next Thursday," might tag "next Thursday" as a date with the standard value 18 July 1991 or, in ISO format, 1990-07-18);
- tags for recording editorial interventions, such as corrections in the text, normalized spelling, additions, deletions;
- page and line references to canonical editions;
- text-critical apparatus; and
- most notably, a set of tags for the specification of linguistic analysis or other interpretive material relating to a text, which can be used (for example) to specify part of speech information or syntactic structure for every word or sentence of a corpus.

This wealth of annotation markup reflects, of course, the particular interest in analysis and interpretation of existing texts found in the research community, the need for which led to the creation of the TEI as a project.

In all, LaTeX and the TEI tag set present a fundamentally similar view of the major components of text; they have much the same view of basic text structure and provide similar facilities for handling most of the phrase-level markup needed for prose. They differ in the amount of attention paid to figures, tables, and similar matter; in the amount of

detail possible for the typographic description of the text; and in the richness of their facilities for annotation. These differences reflect in part the difference between those interested in technical documentation on the one hand (which I take to be the original audience of LaTeX) and those interested in the study and analysis of existing texts, which is the constituency of the TEI. In part, they reflect the difference between a mature piece of software aimed at a particular kind of processing, and a markup scheme still in progress designed to be independent of any particular application and any particular piece of software; and in part, these differences reflect a slightly different model of what text is. It is to this difference that I now turn.

Models of Text and Text Grammars

Any markup language must embody some idea of what text is, formally. How complex and how suitable that idea is for formal processing vary, of course.

Some languages (especially early ones) equate *text* with internally unstructured strings of characters; often this unstructured character string is punctuated by occasional processing instructions that themselves are constrained only by specific implementation details. When no processing instructions are allowed, you have *ASCII-only text*, in which markup is limited to the command repertoire of a 1956 Teletype machine (carriage return, vertical and horizontal tab, backspace, and bell).

For serious processing, extensive command sets have been developed, mostly oriented to the task of getting ink on paper in the right places. Commonly known schemes of this type include Waterloo and IBM Script, troff, most word processors, and, of course, TeX. Processors built on this model are flexible and very easy to understand, but very difficult to prove correct. The number of states in which such a processor can be explodes with the number of commands, and there can be very tricky interactions among various states. Since the state of the system at any point is a function of the entire document up to that point, it is hard to process documents in languages like this except by left-to-right scanning. And since almost any string of characters and commands is legal, these languages offer no real help in verifying the structural validity of machine-readable documents.

A dramatic reduction in the combinatorial explosion of possible states comes with systems that view text as a block-structured hierarchy. The hierarchy is typically a relatively simple one of two or three levels, below which one is back in a sort of primordial prose soup without visible structure. Well-known markup languages in this class include IBM and Waterloo GML, various macro languages for Script and troff, some style packages for microcomputer word processors, and, of course, LaTeX.

These languages introduce a new (hierarchical) model of text, and can thus avoid some interactions among states by simply declaring them illegal. Thus, in LaTeX, it is not legal to have a document body without an enclosing document environment, and, in Waterloo GML, the software checks to ensure that the front matter does not follow the back matter. But no formalisms are introduced to make the document hierarchy fully explicit; there is no explicit *document grammar*. It is naturally impossible then to enforce document validity fully or automatically. Waterloo GML does not check to see that the back matter does not precede the body of the document. Since the more rigid notion of valid document structures is not consistently enforced, these document languages are a bit like programming languages with weak type systems and automatic type coercion and control structures built around the *GOTO*, by relying on the user to follow *good practice* rather than by verifying that good practice formally and mechanically. The constraints which are enforced are hard-coded into the processing code and can thus be hard to change.

The next distinct model of text visible in text processing uses fully explicit, well-defined hierarchies of text elements to define legal text structures. In some cases, like Word Cruncher markup, the hierarchy is so simple that there may still be no explicit specification of the underlying document grammar; in others, the legal structures of documents are specified explicitly and can thus be enforced formally. The best-known markup scheme in this class is the Standard Generalized Markup Language (SGML), which differs from its prototype (IBM GML) precisely in the addition of explicit document grammars with context-free power. (Strictly speaking, of course, SGML is not a markup language but a meta-language that allows the definition of markup languages, precisely because it provides an explicit language for the expression of document grammars.)

SGML markup is of two types: Structural units of the text or specific points in the text (*elements* in SGML parlance) are indicated with SGML *tags*, delimited conventionally by angle brackets or by left-angle-bracket-plus-slash and right-angle bracket. Segments of the text are delimited by a *start-tag* and an *end-tag*, much the same way structural units in LaTeX are delimited by `\begin{environment}` and

`\end{environment}` commands or by left and right braces.

The second type of SGML markup, *entity references*, allows one to insert characters in a document by referring to an *entity* that contains those characters. Entity references can thus be used for special characters not on one's keyboard (analogous to LaTeX's commands for accented letters, etc.), for include boilerplate language (analogous to user-defined macros in TeX that insert formulaic language into a document), and to include external files (analogous to LaTeX's `\input` and `\include` commands).

Any markup used in an SGML document must be explicitly *declared* in a *document type declaration*. Entities are declared by specifying their name and the replacement value (which can be the name of a system file or a string of characters). Elements are declared by specifying their name and their allowable *content*; the declaration for element X specifies what can occur within an X (or within the scope of an X tag), such as character data, other tags, etc. The document type declaration is thus similar to a grammar that specifies the legal forms of a document of a given type; the individual declarations correspond to the production rules of a grammar in Backus-Naur Form (BNF).

The SGML element declaration, however, uses a slightly richer notation than BNF. The *content model* of an element is (more or less) a regular expression composed of the names of SGML elements and the keyword *#PCDATA* (*parsed character data*). SGML thus resembles a regular right-part grammar more than BNF does, but there are further wrinkles we need not go into here that can make SGML content models slightly more compact than regular right-part grammars.

The use of an explicit grammar, together with the explicit delimiters for enclosing each SGML element, leads to a natural view of an SGML document as a tree rather than as a simple unstructured string. The complexity of the processing is contained, since the grammar is basically context-free, and the state of the system at any point in the text can be read by traversing the tree from the root node. LaTeX documents (like any documents with a block-structured model of text) can be treated this way, but the absence of any explicit grammar tends to make such treatment a purely academic exercise.

Specific Design Issues

Some design issues arise in any attempt to specify a document structure that is at once rich and flexible enough to be usable in practice and rigid and precise enough to be formally verifiable.

Prescription and description. First of all, one encounters a fundamental tension between prescriptive and descriptive specifications of document structure. If one is purely prescriptive, one can ensure that the documents one processes will all have very similar structures. Software can make good use of this consistency. However, when one is encoding an already existing text written by someone else — posssibly long dead — it is fruitless to expect it to match a specific prescriptive document style, and historically misleading to try. Rigid formal document specifications may fail to match the chaotic reality of historical documents; unless we are willing to violate the historical integrity of the texts we are studying, we have to provide a more flexible formal structure within which we can find a representation even for unconventionally structured texts.

Excessive flexibility means, of course, that the document grammar may allow spurious document structures that never would occur in practice. Given the choice between excessive rigidity, which makes some documents unrepresentable unless the grammar is loosened, and excessive flexibility, which makes some errors undetectable unless the grammar is tightened, the TEI has consistently chosen excessive flexibility. The issue does not arise in this form for LaTeX, because it does not claim to provide a markup language for arbitrary existing texts; it is comfortable, therefore, with its current degree of prescriptiveness.

Controlling complexity through modularity. Whenever a document grammar is rich enough to handle real texts with serious markup problems, it has enough markup primitives to begin confusing users and developers. It is useful, in this case, to group tags into tag subsets that can be defined and understood independently of each other; this helps control the overall complexity of the markup scheme. Of course, it helps a lot if the software can guarantee that tags in different subsets don't have long-distance interactions. We can see such modularity in LaTeX in the separation of the specialized tags needed for slides and bibliographies into the semi-detached units of SliTeX and BibTeX. In the TEI, similarly, the tags for specialized uses are entirely separate and have no interaction with the core tags for phrases and the like. Linguistic analysis, text criticism, editorial intervention, etc., can all be turned on or off by the user. The current direction of development will lead to more such encapsulations in the next version of the TEI DTDs.

The user, of course, may need to use arbitrary combinations of these specialized tag subsets together; this requires a careful specification of their semantics to avoid side effects.

Multiple hierarchies. Although most texts fall comfortably into a hierarchical analysis of their parts, the use of cleanly hierarchical, block-structured markup does lead to problems whenever the text falls comfortably into more than one such hierarchical structure. The volume, page, column, and typographic line numbers of a standard edition form a simple, clean hierarchy, but one which probably does not nest well with the logical hierarchy of part, chapter, section, paragraph, sentence, and word. If there are several standard editions whose page references should be noted, we have one hierarchy for each edition. When the text is in verse, we can add the metrical hierarchy of canto, stanza, line, and foot. And, of course, the labors of scholars may assign rhetorical, thematic, narrative, and other structures to the text.

The TEI scheme uses the SGML feature of *concurrent markup* to allow the user to maintain several hierarchies in the same document. Bound by the strict block structuring of TEX, it is hard to see any solution to this problem for users of LATEX except to choose one hierarchy as the main one, and to reduce the other hierarchies to simple scope-less declarations in the text.

Systemic comparison of SGML and LATEX. LATEX and SGML resemble each other strongly in their common goals of providing system- and device-independent markup and processing for texts, and in their basically similar hierarchical models of text. SGML pushes the hierarchical model and the notion of formally specified, verifiable document structure farther than does LATEX. It provides a mechanism for formal specification of a document grammar, and validates the document automatically against that grammar.

SGML attempts to provide a notation that is not only system- and device-independent but also software- and application-independent. The origins of SGML are in attempts to ensure the reusability of machine-readable texts by divorcing markup from processing, and stressing descriptive or logical markup rather than procedural markup. LATEX stresses the utility of logical markup to ensure the structural soundness of a document and to make it easy to lay it out in different styles. SGML and the TEI push that concept farther and stress the importance of logical markup in ensuring that a document can be processed without change for entirely different applications, including applications that have nothing to do with text layout or typesetting.

This insistence on application-independence leads SGML into what is its most striking feature as a markup language: its complete lack of semantics. SGML markup languages are entirely declarative, not least because SGML simply provides no formal mechanisms for defining any non-declarative meaning for them. SGML allows you to say that a given stretch of your document is (say) a quotation. It does not require that your say how you want it processed; indeed, it makes it impossible for you to do so in SGML. You specify how an application program should process an SGML document by talking to the application program, not by talking to SGML. The document itself remains a logical object untouched by specific processing instructions. (N.B.: Inserting processing instructions directly into SGML documents is allowed, provided the instructions are explicitly marked as processing instructions so they can be skipped by other software.)

Coexistence

The TEI is intended to be an application-independent markup language for texts of any period, any genre, and any language. Because many of its users will need or want to use already existing software for processing their texts, without modifying that software to read SGML documents, the TEI is intended from the outset to coexist with other software-dependent file formats. The fundamental similarities of goal and the basic harmony of their common emphasis on the logical structure of text combine to make it very simple for the TEI scheme to coexist with LATEX in a single system.

Any file stored locally is stored in some particular file format. This *local storage format* may or may not be identical with the input format of any application program. If only one application is run on it, the file is almost certain to be stored in that application software's input format. A document processed repeatedly with several different packages, however, might have its own format, from which it is translated into the input formats required by the software.

One obvious use for a scheme like the TEI tag set is as a local document storage format. When one wants to make a concordance from a document, one translates it from the TEI format into the form required by the Oxford Concordance Program or some other concordance package; when one wants to make

hard-copy output, one translates it into the form required by the desired formatting or typesetting program. The structural similarities of the TEI scheme and LaTeX mean that a TEI-to-LaTeX conversion is relatively straightforward, and for the most part the same may be said of a LaTeX-to-TEI translation.[1] In other words, LaTeX is a natural choice for the typesetting of TEI-tagged documents, just as the TEI format is a natural choice for the encoding of a text's logical structure so that it can be processed by many different pieces of software.

Acknowledgments

The TEI, an international cooperative effort to develop and disseminate a common format for the encoding and interchange of machine-readable texts, is sponsored by the Association for Computers and the Humanities, the Association for Computational Linguistics, and the Association for Literary and Linguistic Computing.

It is funded in part by the U.S. National Endowment for the Humanities, an independent federal agency; DG XIII of the Commission of the European Communities; and the Andrew W. Mellon Foundation.

The work is done by many generous individuals from the community who volunteer their time to serve on the working committees and work groups of the project.

References

[1] DeRose, Steven J. "Suggestions for improving the AAP tag set." TEI working paper, document TEI TR R7, August 1989.

[2] Knuth, Donald. *The TeXbook.* Reading, Mass.: Addison-Wesley, 1984.

[3] Lamport, Leslie. *LaTeX: A Document Preparation System.* Reading, Mass.: Addison-Wesley, 1986.

[4] Sperberg-McQueen, C. M., and Lou Burnard, eds. *Guidelines for the Encoding and Interchange of Machine-Readable Texts.* Text Encoding Initiative, Chicago, Oxford, draft version 1.1 edition, 1990.

[1] This document, for example, was drafted using SGML tags and converted to LaTeX for submission.