ELECTRONIC PUBLISHING, VOL . 2(1), 3–24 (APRIL 1989)

Why use SGML?

DAVID BARRON

Department of Electronics and Computer Science
University of Southampton,
Southampton SO9 5NH, UK

SUMMARY

The Standard Generalised Markup Language (SGML) is a recently-adopted International Standard (ISO 8879), the first of a series of proposed Standards in the area of Information Processing — Text and Office Systems. The paper presents some background material on markup systems, gives a brief account of SGML, and attempts to clarify the precise nature and purpose of SGML, which are widely misunderstood. It then goes on to explore the reasons why SGML should (or should not) be used in preference to older-established systems.

KEY WORDS SGML Markup Generalised markup Formatters

INTRODUCTION

The Standard Generalised Markup Language [1, 2, 3] (henceforth referred to as SGML) is one of an extensive collection of proposed International Standards in the area of office automation, originally categorised by the International Standards Organisation (ISO) as Computer Languages for the Processing of Text (CLPT), and later re-named "Information Processing — Text and Office Systems". In addition to SGML the suite of standards will eventually include

- the SGML Document Interchange Format (SDIF)
- the Document Style Semantics and Specification Language (DSSSL)
- the Standard Page Description Language (SPDL)
- a reference model for text description and processing languages
- standards for font and character information interchange
- a set of standard fonts.

The International Standards Organisation is also working on standards for Office Document Architecture (ODA) and Office Document Interchange Format (ODIF) [4]. The relation of these standards to SGML will be explored briefly in a later section.

SGML has taken many years to reach its present state of acceptance, so we should not wait with bated breath for the imminent arrival of the brave new millennium of standardised text processing. (If such it be. I will not explore here the enticing but perhaps mischievous argument that *de facto* standards emanating from public demand such as PostScript and TIFF are to be preferred to standards imposed by an international body of do-gooders.)

SGML has been widely misunderstood, not least by some of its supporters, and has been promoted with almost religious zeal in some quarters — perhaps the most extreme

0894–3982/89/010003–22$11.00 *Received 14 October 1988*
Revised 17 November 1988

example being the claim by Smith [5] that it can be used as a software development tool. This paper therefore sets out to put SGML in perspective: to explain what it is, what it is useful for, what the arguments are for using it, and what the arguments are for not using it. First it is necessary to explore the meaning of the term ‘markup’ to put SGML in perspective.

MARKUP

Markup in the traditional publishing process

Although Coombs *et al* [6] define markup in a very general manner to include inter-word spaces, punctuation and layout, we shall adopt the more restrictive (but more conventional) definition given in *The Chicago Manual of Style* [7], where markup is defined as

“The process of marking manuscript copy for typesetting with directions for use of type fonts and sizes, spacing, indentation etc.”

In the traditional (pre-electronic) publishing process, this process of ‘marking up’ the manuscript was a major pre-press function of the copy editor: the book designer decided all the formats and layouts, and the copy-editor translated these into detailed instructions for the compositor, written on the manuscript. For example, at the start of a chapter we might find “10/12 Times Roman × 24”, indicating that the chapter was to be set in 10-point Times Roman on 12-point leading, to a measure of 24 picas (printer’s jargon for a line length of 6 inches): at the start of a block quotation the copy editor might write “10/10 Times Roman indent 2 ems from left” to indicate that the quotation was to be set solid with an indentation from the left margin. A coloured vertical line to the left would indicate the exact extent of this different setting.

Electronic markup

In the preceding section, ‘markup’ was defined as a process applied to a manuscript. In the world of computer text-processing the term is commonly used as a noun to describe any codes or instructions included in a document to specify formatting and layout, by analogy with the written annotations added to a manuscript by the copy editor to direct the compositor in setting the text. All word processing and desktop publishing systems include some markup in this sense: even in a WYSIWYG system, where formatting happens on screen in response to a function key or menu choice, the system must insert additional codes in the file at the point in question so that the effect can be recreated when that part of the file is next displayed, or is printed. This markup is necessary to flag changes of type-style (e.g. bold, italic, etc.) at the very least. For example, when a user of WordPerfect requests bold type, the system inserts an 8-bit extended-ASCII character in the file as a control code, at the same time changing the colour of the screen display. This corresponds very closely to the wavy underline and the marginal note ‘bold’ of traditional copy-editing. In early WYSIWYG systems (e.g. the original WORDSTAR) this kind of markup was visible on screen, but in modern systems it is usually concealed from the user in normal operation. There may be an option to display the markup: for example, if the WordPerfect user switches the ‘reveal codes’ option on, the codes are displayed on-screen as meaningful words enclosed in square brackets (though as noted above the markup is in fact encoded in the file as a single character from the extended ASCII character set.)

Markup at this level is associated with the particular point in the file at which it occurs: more recent systems (e.g. Microsoft WORD) have adopted the use of style sheets, which are the electronic equivalent of the book designer’s general rules. Each paragraph is tagged with a code that determines the style in which it will be laid out. (For this purpose a section heading, for example, is defined as a paragraph). The tags are usually concealed from the user, being applied by a key combination (Microsoft WORD) or by clicking a menu item (Ventura Publisher): as an alternative, in Ventura tags can be entered as ASCII strings in the input file in the same form as they are stored in the file by the system when invoked by menu choice. These tags are a first step towards what Coombs *et al* refer to as *descriptive markup*, since they can be thought of as describing the function of each tagged element, e.g. first-level heading, block quotation, normal paragraph, etc. However, they are normally regarded by the user as a convenient shorthand for a description of a visual layout.

Batch formatters and generic markup

Our concern in this paper is not primarily with the hidden markup of WYSIWYG systems: it is more with the explicit markup that one encounters in ‘batch formatters’ such as troff [8], TEX [9], SCRIPT/VS [10] etc. These follow a paradigm first used in the archetypal RUNOFF utility in CTSS [11, 12]: formatting commands are interspersed in the text, being identified by some characteristic syntax that is unlikely to occur in running text. In the original RUNOFF, formatting commands were recognised by the presence of a period at the start of a line, and this convention was adopted by all the runoff programs descended from the CTSS original, in particular troff. A later feature of troff was the embedded request (i.e. a request not necessarily at the start of a line) introduced by a backslash, and when Knuth devised TEX he made all commands embedded, using backslash to mark a command word; this makes human reading difficult, but machine parsing easy.

The formatting commands of these systems can be called *procedural markup* since they are all concerned with the appearance of the text, controlling fonts and spacing at a very fine level of detail. This is not a level at which it is comfortable to work, and as early as 1976 troff was enhanced by a set of macros, the classic *ms* macros.[1] These macros abstracted the structure of the document, allowing the troff user to mark the major components of a document — paragraphs, headings, lists etc. — without worrying about the typographical rules for the layout of the various components; these were encapsulated in the macro definitions, and so not accessible to the general user, who was insulated from this level of detail. Uniformity of appearance was guaranteed; the Bell Laboratories Technical Reports illustrate this uniformity well. In 1977 Knuth started work on TEX, which became available in its first version in 1979. From the start TEX had a collection of macros (plain TEX), but these were of the nature of procedural markup, and it was not until comparatively recently that Lamport developed a set of

[1] The ms macros have been superseded in Unix System V by the *Memorandum Macros*, (*mm*): these provide increased functionality, but the principle is the same.

macros with a similar functionality to ms, which separate the logical structure of the document from its appearance. The resulting LaTeX system [13] has an almost cult status amongst many academics. (Another widely-used package in the TeX community is AMS-TeX [14], a macro package developed for the American Mathematical Society to do mathematical typesetting. However, AMS-TeX mainly provides a specialised form of procedural markup for mathematics — equations and formulae are not well adapted to generalised markup.)

These systems are a closer approach to Coombs's descriptive markup, though they do not meet all his criteria — there are some aspects of formatting that still require procedural markup. They are in fact instances of *generic markup*. This is a term coined in the printing industry to describe the technique of identifying major components of a document — paragraphs, headings, quotations etc. — with tags that describe their function in a form independent of any particular typesetting machine. The macro calls are analogous to the paragraph tags of Microsoft WORD or Ventura, and the macro definitions are the equivalent of the style sheet (except in regard to comprehensibility.)

Generalised markup

At the same time that troff/ms and TeX were being developed, a group at IBM led by Goldfarb was developing text processing tools. The separation of structure from appearance, advocated by Reid [15] and implemented in the SCRIBE system [16] at an early stage had also motivated the IBM group, and it is to Goldfarb that we owe the term "Generalised Markup" to describe this approach to document preparation. His 1981 definition [17] cannot be bettered:

"...it does not restrict documents to a single application, formatting style, or processing system. [It] is based on two novel (at the time) postulates:

(1) Markup should describe a document's structure and other attributes, rather than specify processing to be performed on it, as descriptive markup need be done only once and will suffice for all future processing.

(2) Markup should be rigorous, so the techniques available for processing rigorously-defined objects like programs and databases can be used for processing documents as well."

Goldfarb's ideas were embodied in the "Generalised Markup Language" which was adopted by IBM and now forms part of their flag-ship text processing system DCF/GML (Document Composition Facility – Generalised Markup Language) [18]. The terms 'generalised markup' and 'descriptive markup' are largely interchangeable. Some clue to the significance of the term 'generalised' can be found in Goldfarb's observation that markup should describe a document's structure *and other attributes*. For example, we shall later encounter an instance where markup is used to code semantic content in a document.

The benefits of generalised markup

The benefits of generalised (descriptive) markup have been rehearsed extensively elsewhere (see for example Smith [19] and Coombs *et al* [*loc. cit.*]), and we summarise them very briefly here. The essential feature is that generalised markup completely divorces structure from appearance: the author uses markup to describe the structure of his document, and to specify his *intent*, without regard for appearance. Thus if he wants a phrase emphasised it should be described as ‘emphasised’, not as italic. (If the surrounding text is in italic, the emphasis will normally be conveyed by using Roman type for the phrase.) To take another example, a word-processing style sheet might define a paragraph format for a display, set off from the surrounding text by extra spacing and indentation; this could then be used for a long quotation, or as a way of highlighting some text. In a generalised markup system we would tag each according to its function, quotation or highlight. The actual form in which each will appear is defined quite separately, and although one system might choose to use the same block display for both purposes, another might differentiate them, possibly setting quotations in a different typeface to make them stand out from the running text. Equally important, tagging in this way facilitates indexing and the generation of selective ‘views’ of a document, e.g. a collection of all the quotations. Another example comes from the use of quotation marks in running text: publishers have differing conventions regarding the use of single and double quotes, and so it is important that quotations should be identified as such without prejudging the kind of quotation marks to be used. (de la Beaujardiere [20] claims that quotation marks are used for three quite distinct purposes: to delimit verbatim quotations, to denote irony, or to indicate a nick-name, and that in a generalised markup system the author can use different tags to signify his intent in these situations. Whilst this is undoubtedly true, the example seems contrived.)

Generalised markup can also facilitate other aspects of document production. For example, in a textbook we might tag each technical term on its first appearance: this might not only cause it to be printed in a distinctive typeface, but also helps in the process of compiling an index of definitions. In a collection of poems the first line of each poem might be specially tagged, not so that it could be printed differently but solely to simplify the process of creating the index of first lines. In a complex literary work (e.g. *The Lord of the Rings* [21]) we might tag the names of the principal characters every time they appear in the text with a view to creating a concordance for scholars who wish to compare the context in which characters appear.

In summary, the use of a generalised markup system forces the author to pay attention to the structure of his document, whilst giving the publisher control over the appearance and facilitating the enforcement of a house style. Even if the author and the publisher are the same person, it is good practice to concentrate on the content and structure of the document at the writing stage, and not to be distracted by presentation issues.

MISCONCEPTIONS ABOUT SGML

There are a number of formatting systems around that implement a form of descriptive markup: IBM’s GML, $\LaTeX$, and the mm (or ms) macros with troff are the most commonly used. Each of these systems forms a *de facto* standard within a particular community of users, and the benefits of using a formatter based on a descriptive markup system are so compelling that there are persuasive arguments for a single standard markup language.[2] The first misconception (and a very common one) about SGML is that it is such a standard system. This is hardly surprising: it is after all called the

[2] Superficially persuasive: it can be argued that the universal markup language is a chimera of the same ilk as the universal programming language.

Standard Generalised Markup Language. It is in fact a meta-language which can be used to define an arbitrary number of markup languages in a standardised way. The second misconception is that SGML is a formatter, like troff or LaTeX. It cannot be too strongly stated that *SGML is not a formatter*. The belief that SGML is a formatter based on descriptive markup like LaTeX or GML pervades much of the writing on the subject. As an example, in a recent issue of *The Seybold Report on Publishing Systems* a letter to the editor describes a so-called SGML system that "has the ability, by setting a software switch, to process either monospaced or proportionally spaced text", and in the lead article in the same issue Becker [22] describes SGML as "belonging to a class of document development systems like Scribe, troff and TeX", and asserts that SGML is "a meta-language for generating descriptive mark-up languages with a coherent and unambiguous syntax *that allows users to describe how any element of a document should appear in final form*" (my emphasis).

SGML DEFINED

SGML is not a standard markup language, and it is not a formatter: it is something altogether different. SGML is a meta-language that defines only the *syntax* of a standard generalised markup language, i.e. is prescribes *how* we should specify markup, but not *what* that markup is, nor what it means. SGML defines an abstract syntax for a markup language, and so provides a standard mechanism for generating a family of descriptive markup languages which can be used to describe the structure of a document, but nothing else. A descriptive markup language generated by SGML is normally used as a front end to a program that will perform subsequent processing of the document: this may be a formatter if we are producing printed documents, but may just as well be a database management system or a hypertext stack generator.

Within the abstract syntax of SGML there is defined a standard way of specifying the *document type definition* (DTD), which defines the logical structure of a document in terms of the elements that comprise it (paragraphs, headings, footnotes etc.) and their relationships (i.e. the constraints on their appearance — a second level heading can only occur within the scope of a first-level heading, for example). It also associates a *generic identifier* with each element, thus defining the *tags* that will be used for the descriptive markup of a document. In addition the DTD can define the circumstances in which tags can be omitted, their presence being implied by the context (e.g. the start of a paragraph implies the end of the preceding paragraph), and the extent to which the presence of a tag can be inferred from the physical layout of the input document (e.g. the use of a blank line to separate paragraphs as in LaTeX). Finally, SGML specifies a 'reference concrete syntax' which specifies the particular keyboard characters to be used to enter markup in a document, and a way of redefining these characters.

An SGML document is processed by an SGML *parser*[3]. This program checks that the document conforms to the specification laid down by the DTD, and inserts tags whose presence is implied. That is all: the end product is still an SGML-tagged document. What happens next depends on the application. If the end-product is a printed document the output from the SGML parser will usually be passed to another program which maps

[3] This is the terminology firmly established in the SGML community. However, a computer scientist would recognise the SGML processor as a *parser generator* or *compiler-compiler* which takes a formal specification of a language (the DTD) and generates a parser for that language, which in turn is used to process the user's document.

the SGML tags onto the command set of the underlying formatter or composition system. This mapping is totally outwith the scope of SGML[4] (and indeed of any descriptive markup scheme).

The reader may reasonably ask what is the use of a standard that defines the form of the tags but does not prescribe what the tags actually are, nor what their effect is to be. The concern at this level is with the transferability of documents between machines. Using SGML a document with markup can be transferred across a network and the recipient can distinguish unambiguously between document content and markup. However, the document is presumably being transferred so that it can be processed, and therefore there must be agreement between sender and recipient as to the actual tags used, and their significance in the document structure: this is achieved if both use the same DTD. (In some applications it will be necessary that both parties agree as to the mapping of the tags onto actual layout, but this is not implicit in the use of SGML, which allows the sender to convey the structure of the document in terms divorced from particular layout conventions.) It is recognised that documents are too varied for there to be a single international standard DTD, and it is envisaged that a mechanism will be provided whereby DTD's can be registered for public use. Two such DTD's suitable for general use have been defined, one by the British Library [23, 24] (the so-called 'starter set') and one by the Association of American Publishers [25]. If a document conforms to a public DTD, then all is needed is an indication at the start of the document to say which DTD is to be employed. Alternatively, the complete DTD can be prefixed to the start of the document before it is transmitted.

A BRIEF TOUR OF SGML

In this section we present the main features of SGML. It is not possible in the space available to do more than convey the flavour of SGML: reference [24] gives an end-user view of using SGML with a standard DTD, and reference [3] provides a readable and comprehensive view of the inner workings of SGML and the facilities provided to the document designer. Figure 1 illustrates many of the features of SGML: it shows the beginning of this paper as it would appear if we were using SGML with the British Library Starter Document DTD (the 'starter set' tags) and using the default syntax (the reference concrete syntax).

Markup

In the reference concrete syntax angle brackets are used to distinguish markup from the text. (Clearly, if angle brackets delimit markup, they cannot also appear as part of the text: we shall see later how this apparent impasse can be circumvented if we actually want angle brackets in the text.) Most of the markup consists of tags identifying the elements in the document structure, but it also includes *declarations*, i.e. information directed to the SGML parser that will affect the way in which the document is processed, introduced by the sequence '`<!`'. The `DOCTYPE` declaration in the first line of Figure 1 identifies a public DTD to which the document conforms. The parser will retrieve the

[4] Or so one would have thought. However, the latest thinking of ISO Working Group 8 (the group responsible for SGML) is that the semantics that tell text formatters exactly how the text is to be formatted can themselves be expressed in SGML. An exposition of this complex proposal is given by Bryan in Chapter 9 of reference [3].

```
<!DOCTYPE sd PUBLIC
  "+//British Library//DTD Starter Document//EN">
<!ENTITY SGML "Standard Generalised Markup Language">
<sd status="first draft">
<ti>Why use SGML?
<au>David Barron
<ad>
<l>Department of Electronics and Computer Science
<l>University of Southampton,
<l>SOUTHAMPTON SO9 5NH</ad>
<ab><p>The &SGML; (SGML) is a recently-adopted
International Standard (ISO 8879),
...
to explore the reasons why SGML should (or should not)
be used in preference to older-established systems.</ab>
<h1> INTRODUCTION
<p> The &SGML; <r id=SGML> ISO8879, <pt>Information
Processing&en;Text and Office Systems&en;&SGML;
(SGML)</pt></r> (henceforth referred to as SGML) is one of
an extensive collection of proposed International Standards
...
re-named <sq>Information Processing&en;Text and Office
Systems</sq>. The suite of standards will eventually include
<ul>
<li>
the SGML Document Interchange Format (SDIF)
<li>
the Document Style Semantics and Specification Language
(DSSSL)
...
<li>
A set of standard fonts
</ul>
<pc>  in addition to SGML.
The International Standards Organisation ...
```

Figure 1. Example of text marked up in SGML

DTD (presumably stored on disk) and will process it before proceeding further. Thus the effect is as if the whole DTD had been placed at the head of the document. Line 3 of Figure 1 illustrates another kind of declaration, the *entity declaration*: this is discussed further in a later section.

Tags

As will be seen from Figure 1 tags can appear anywhere on the line, though we often place tags alone on a line so that they stand out and give visual clues to assist in proof-

Table 1. Summary of tags used in Figure 1

Tag	*End-Tag*	*Meaning*
<ab>	*</ab>*	*Abstract*
<ad>	*</ad>*	*Address*
<au>	–	*Author*
<h1>	–	*First-level heading*
<l>	–	*Line of address*
<li>	–	*List item*
<p>	–	*Paragraph*
<pc>	–	*Paragraph continuation*
<pt>	*</pt>*	*Publication title*
<r>	*</r>*	*Reference*
<sd>	*</sd>*	*Start of document*
<sq>	*</sq>*	*Short in-line quotation*
<ti>	–	*Title*
<ul>	*</ul>*	*Unordered list*

reading the original document. The tags used in the example are fairly self-explanatory, but for completeness they are summarised in Table 1. A complete description of the BL Starter Set tags is given in references [23] and [24]. Strictly, each element of the document is introduced by a start-tag and ended with a matching end-tag, e.g. `<ab>` to start the abstract, and `</ab>` to end it. However, an optional feature in SGML allows the document designer to specify in the DTD that the presence of end-tags can be implied by the context: in Table 1 a dash in the 'end-tag' column indicates that the end-tag is not required explicitly. Thus the tag `<ti>` which opens the title element implies the presence of a closing tag `</au>` to close the preceding element (author), and in the list introduced by `<ul>` the tag `<li>` that introduces a list item also implies an end-tag for the previous list item. If the parser implements this feature, it will insert the implied end tags into the document before outputting it. This makes the task of mapping the SGML tags on to the underlying formatter easier.

Note the use of `<sq>` and `</sq>` to bound an in-line quotation: as observed earlier, this leaves the publisher free to decide what kind of quotation marks to use.

Attributes

The DTD can specify that a tag has optional or obligatory *attributes*. Two examples of this can be seen in Figure 1. On line 3 the `<sd>` tag has an optional attribute to indicate the status of the document. What use is made of this will depend on the formatter: it might for example be included in the running header when the document is printed. An example of an obligatory attribute is seen on line 17 where the tag `<r id=SGML>` uses an attribute to attach an identifier to a reference, which is given in full. (The formatter will presumably insert a citation in the text at this point and collect all the references at the end of the document.) This identifier can be used to cite the same reference elsewhere in the document e.g.

```
...in the SGML standard.<rr rid=SGML>
```

Here the cross-reference tag `<rr>` has an obligatory attribute to identify the particular cross-reference. This makes cross-referencing independent of pagination. However, this method of dealing with references is simplistic, and ignores many of the problems that arise in real documents. See Rahtz [26] for a comprehensive discussion of these problems, and the way they are addressed by the Unix tool ‘refer’ and the LaTeX-BibTeX combination.

Entities

An *entity* in SGML is a named object. It can be a character string, a special character, or even a file containing part of a document: whatever it is will be identified in the text by a name in the form of an *entity reference*. Line 3 of Figure 1 is an example of an *entity declaration*, in this case associating the identifier ‘SGML’ with the string “Standard Generalised Markup Language”. This long string which occurs several times in the subsequent text can be abbreviated by the entity reference ‘`&SGML;`’ whenever it is required. The ampersand and semi-colon are the delimiters of the reference; the characters in between name the entity that is being referenced. Many entities are pre-defined in a typical DTD: for example, on line 18 of Figure 1 we see the sequence ‘`&en;`’ used to signify an en-dash.

Here we have used an entity reference to indicate a character that is not available on the normal typewriter keyboard. Entities are frequently used in this way for characters that are not in the ASCII (or more precisely ISO-646) character set, e.g. accented letters, non-Roman alphabets, mathematical symbols etc.). Using entities makes references to these characters independent of the properties of a particular typesetting system. Another use for entity references is to name files that contain sections of a document. As with the simple text string, the entity reference is replaced by its definition whenever it occurs, so that using it to name a file provides the kind of ‘include’ facility that is commonly found in text processing systems.

Finally, entities resolve the problem of including in the text characters that have a special significance for the SGML parser, e.g. the angle brackets that enclose markup. For example, the declaration

```
<!ENTITY lt "<" >
```

allows us to include a left angle-bracket as ‘`&lt;`’. (Although this is a neat way of using an existing concept to solve the problem, users might prefer the troff and TeX solution of using an escape character to remove special meanings from meta-symbols: if the left angle-bracket were a meta symbol in either of those systems, a literal left angle-bracket would appear in the text as ‘\<’. This requires fewer keystrokes, and makes the text more readable.)

Defining the document structure

We have seen that the document structure is defined in the DTD which defines the tags and their relationships. Here we give just a simple example to give the flavour of a DTD, taken from Annexe A of the SGML Standard [1].

The aim is to define markup for a figure that might appear in a technical document. The figure body may consist of artwork or text (which may include lists), and the figure may have an optional caption. The tag that introduces the figure may include an optional identifier attribute, so that the figure may be referenced from elsewhere in the document,

```
<fig id=babel>
<figbody>
<artwork depth=3in>
<figcap>The Tower of Babel by Pieter Brueghel (1563)
</fig>
```

Figure 2(a). SGML markup for a figure

```
<!ELEMENT fig      --  (figbody, figcap?)>
<!ELEMENT figbody  -O  (artwork | (p | ol | ul)+)>
<!ELEMENT artwork  -O  EMPTY>
<!ELEMENT figcap   -O  (#PCDATA)>

<!ATTLIST fig      id     ID     #IMPLIED>
<!ATTLIST artwork  depth  CDATA  #REQUIRED>
```

Figure 2(b). SGML declarations for figure markup

and if the figure consists of artwork the appropriate tag *must* include an attribute specifying the size, so that the formatter can leave an appropriate gap in the text for the figure to be pasted in.

Figure 2(a) shows an example of the use of the tags, and Figure 2(b) shows the SGML declarations necessary to define them. In Figure 2(b) we see first declarations of the elements to be tagged in a figure. Line 1 declares an element *fig* (and hence the associated tag `<fig>`) and asserts that it consists of an obligatory *figbody* and an optional *figcap*. The two dashes indicate that both open and close tags are required for a *fig*. The next line defines a *figbody* as either artwork or an indefinite number (at least one) of occurrences of a paragraph, ordered list or unordered list. (It is assumed that there are already definitions for paragraph (`<p>`), ordered list (`<ol>`) and unordered list (`<ul>`)). The characters '`-O`' signify that the start-tag is required, but that the end-tag can be omitted so long as its presence can be unambiguously inferred from the context. Line 3 specifies that *artwork* has no content (it is something that will be provided outside SGML), and line 4 defines *figcap* as an arbitrary string of characters. (PCDATA indicates that the string will be parsed by the SGML parser, and therefore may include entity references.) Finally we have two declarations that specify attributes to tags. The first of these says that *fig* has an optional attribute with name 'id', of type ID — a code denoting a unique identifier. The second says that *artwork* has an obligatory attribute named 'depth', whose value is a character string. CDATA indicates that this string will not be processed by the SGML parser.

Public text

Figure 1 illustrated the facility to reference a pre-defined DTD, the British Library 'starter document'. This is an example of 'public text', i.e. a collection of SGML declarations made available for general use and intended as an informal standard. Such collections, do not form part of the Standard *per se*, but it is suggested in the Standard that they should be registered with ISO. Public text is not limited to DTDs: other public text collections include recommended names for characters in non-Roman alphabets, so that e.g. a French e-grave can be referenced as '`&egrave;`', and for mathematical symbols e.g. '`&infin;`' for the 'infinity' sign.

Alternative concrete syntax

The concrete syntax is defined in a declaration introduced by `<!SGML` which must appear as the first declaration in a document. (Normally the SGML declaration is implicit: the parser reads the standard declaration from a file before reading the document, but an explicit declaration can be prepended to the document if required.) It is possible for the expert user to completely redefine the concrete syntax in an arbitrary manner, but this is deprecated. It is expected that a number of alternative concrete syntaxes will be available as public entities; and users will select one of these, sometimes with small local changes. One such alternative is the 'Multicode Basic Syntax' which is defined for use in non-English applications where the repertoire of available characters is extended by shifting in and out of alternative character sets, and it is important that markup should be recognised only in the base character set. Another likely change in the concrete syntax would be to change the tag delimiters. For example, users accustomed to IBM's GML might prefer to have tags delimited not by angle brackets but by colon and period to conform to IBM's convention.

If the SGML declaration is used to define an alternative syntax it must itself be expressed in the reference concrete syntax, for obvious reasons.

Minimization

We have seen that SGML provides the optional capability for the document designer to specify that end tags are implied by context. This is the most common use of *minimization*: SGML provides a number of other short-cuts to reduce the number of keystrokes required in preparing a document. Provided that the parser supports the options, the following minimizations can be specified in the DTD.

- Omission of an end tag. An end tag can be omitted if its presence can be unambiguously inferred. For example, a paragraph does not normally need an end tag.
- Omission of a start tag. A start tag can be omitted if its presence can be unambiguously inferred. For example, in an ordered list the first item does not need a start tag.
- Empty start tag. An empty start tag (`<>`) is treated as a repetition of the most recent start tag, if the ability to omit tags is switched on. If the DTD specifies that tags cannot be omitted, an empty start tag takes the same identifier as the most recent end tag.
- Empty end tag. An empty end tag (`</>`) matches the most recent start tag.
- Unclosed tags. If two or more tags occur consecutively, the end delimiters of all except the last tag in the sequence can be omitted. Thus the sequence `</ul><pc>` at the bottom of Figure 1 could be abbreviated to `</ul<pc>`.
- Null end tag. The end of an element can be marked by a single character, solidus in the reference concrete syntax. This capability is selected on a per-element basis in the DTD: obviously it must be used with care, since the selected character cannot appear in the body of the element.

Automatic tag recognition

The ultimate in markup minimization is no markup. SGML provides for this by the ability to specify that a sequence of characters that forms part of a document is also to terminate an element. This termination may in turn allow the parser to infer the presence of a start tag for the next element using the omitted start tag mechanism. Using this facility it is possible, for example, to define an empty line as a paragraph separator which simultaneously ends one paragraph and starts the next.

Graphics, Tables and Formulae

Technical documentation frequently includes material such as tables and mathematical formulae in addition to text, and there is an increasing requirement to incorporate graphics as well. A table has a regular structure, and it it therefore feasible to define a set of tags to reflect this structure. The British Library Starter Document DTD includes such definitions, and the Association of American Publishers has a parallel set of recommendations [27]. Generic markup is not the only way of describing tabular material: an alternative approach which has much to commend it is to use a special-purpose language for the purpose, such as Unix *tbl* [28]. Herein lies a potential problem for SGML: if the underlying formatter is troff, which expects its tables to be described in tbl notation, the mapping of the SGML tags will be very tricky. It is not always possible to divorce the markup completely from the tool that effects the layout. Mathematical formulae are more intractable since they do not lend themselves readily to descriptive markup. Although the Association of American Publishers has defined a set of tags for mathematical material [29], the most successful mathematical formatting systems (e.g. Unix *eqn* [30], which acts as a preprocessor to troff) use special notations not based on tags. There is a hint in an annexe to the SGML Standard that users would employ one of these well established notations with tags to delimit this ‘foreign’ notation so that it can be passed through unchanged by the SGML parser, as shown in the following example.

```
<p>The formula
<formula notation=EQN>I = E over R</formula>
is known as Ohm’s Law.
```

As in the case of tabular matter, this presupposes that the underlying formatter is capable of processing this notation.

The issue of graphics is neatly sidestepped by a statement that SGML supports ISO graphics standards, particularly the Computer Graphics Metafile (CGM) [31]. In practice, SGML merely provides a way of declaring an element to be “non-SGML data” to be passed through the system untouched. Thus any graphics that can be coped with by the underlying formatter can be handled, not just CGM.

Dirty tricks

Although the philosophy of SGML is that the markup should represent the structure of the document, and be completely divorced from the layout, nevertheless there is provision for including in an SGML document ‘processing instructions’ which are meant for the underlying formatter. In the reference concrete syntax the form is `<? ... >`: the parser on encountering this construct will pass the text up to the closing delimiter to the output without examining it.

THE MANY FACES OF SGML

The foregoing should have convinced the reader that SGML is large and complex: indeed, we have not even mentioned the more complex and esoteric features such as concurrent document structures, where the DTD defines two possible structures for the document, and the means of mapping between them. SGML can be viewed in many ways, and here perhaps is another source of confusion: it can appear as one of a number of rather different systems, depending on the context and the point of view.

At one extreme we might want to use SGML simply to identify markup unambiguously, so that it is not confused with the text of the document. A recipient of a document might for example just want to discard the markup: compare the action of saving a word-processor file in plain ASCII format.

More commonly an author might want to prepare an electronic manuscript in a form that could be accepted by any publisher. For this he would use a public DTD such as the British Library Starter Set and type his tags in the concrete syntax (unless he knew the document was to be processed on an IBM system, in which case he would want to use the same tags but with a different syntax to match GML conventions). SGML is not a formatter, so for this application the author would be using a system in which SGML was 'bolted on' to an existing formatter, and it is scarcely surprising that at this level, using SGML is much the same as using any other descriptive markup system. Indeed, it is an almost trivial exercise to write a program to map the tags of the BL starter set on to equivalent constructs in LaTeX or troff/mm format.

At the other extreme we may have to deal with a document that has an extremely complex structure: a good example is a dictionary. Using SGML we can construct a DTD that encapsulates the structure of the dictionary, and the text can then be prepared in machine-readable form with the structure explicitly indicated by the tags as defined in the DTD. We have in effect used SGML to create a special-purpose language to describe dictionaries. Once we have the structure explicitly exhibited by the tags, all sorts of subsequent processing is possible. We can print the dictionary, using the tags to determine layout and typeface, we can abridge the dictionary, using the tags to select things to be omitted, we can put the dictionary on CD-ROM using the tags to construct an index, or we can convert it into database format and use the tags for all manner of searches. An application of current interest is the conversion of such a fully-tagged document into a hypertext document.

By way of illustration, we use the electronic version of the Oxford English Dictionary [32, 33]. Whilst a dictionary might appear to have a very simple structure — it is just a list of words each with an associated definition — it is in fact an extremely complex entity. Some elements of a definition are introduced by special symbols, but most of the structure of the definition is indicated implicitly by position and by subtle use of different typefaces and sizes. In the New Oxford Dictionary, all this structure is captured in the form of SGML tags, as illustrated in Figure 3 (taken from Raymond and Tompa [34].)

Once the text has been tagged in this way, all sorts of possibilities emerge. Different physical formats can be generated. Abridged versions of the dictionary can be generated

```
<entry><hwgp><hwelm> abbreviate </hwelm> <pron id=000041884>
a&breve.br<i>i&mac.</i> &sd.vi&sylab.<i>e</i> <su>i</su> t
</pron> <pos>v.</pos> </hwgp>
<vfl> Also <vd>5&en.7</vd> <vf>abreviate</vf> </vfl>
<etym> f.<xra id=000041880><xlem>abbreviate</xlem> <pos>ppl.
a.</pos> </xra> or on the analogy of vbs. so formed; see
<xra id=000041881> <xlem>-ate</xlem> </xra> &es.A direct
representative of L. <cf>abbrevia&mac.re</cf>; as
<xra id=000041882><xlem>abridge </xlem> </xra> and the obs.
<xra id=000041883><xlem>abrevy</xlem> </xra>, represent it
indirectly, through OFr. <cf>abbregier</cf> and mid.Fr.
<cf>abre&acu.vier</cf>. &es.Like the latter,
<cf>abbreviate</cf>, was often spelt <cf>a-breviate</cf> in
5&en.7. </etym>
<sen4><sen6> To make shorter, shorten, cut short in any way
<qpara><quot><qdat>1530</qdat> <auth>Palsgr.</auth>
<qtxt> I abrevyate: I make a thynge shorte, <i>Je abrege</i>
</qtxt> </quot></qpara></sen6></sen4> ...
```

Figure 3. Dictionary entry with SGML tags

by selectively ignoring certain tagged items: for example, etymologies can be discarded by ignoring text between `<etym>` and `</etym>`, or we can count all the verbs in the dictionary by selecting entries that include the sequence `<pos>v.</pos>`. Scholars can readily access all the quotations used to illustrate etymology. The dictionary can be converted into a database, or into hypertext.

ARGUMENTS AGAINST SGML

It can safely be assumed that the reader who has persevered thus far is convinced of the merits of descriptive markup. The question remains, though, 'why use SGML?'. Although the answer to the question is largely implicit in the discussion in the preceding sections, we explore it in a little more detail here.

Proponents of SGML put forward a number of arguments in its favour, the main one being that a standard markup language facilitates document interchange. We first note that for meaningful document interchange we must also agree on the descriptive markup tags, which are not part of SGML proper. Certainly, the benefits to authors of a universal standard, e.g. SGML with the British Library Starter Set tags, are self evident. A book or journal article could be transferred in machine readable form, being printed out in various formats at different stages in its production. Bibliographic information could easily be extracted by standardised tools. However, to achieve these benefits will require not only the development of new software systems but also changes in the established manner of working of publishers and possibly authors as well. Authors will not in general be willing to tag their documents according to the standard convention, nor can they be relied upon to do it correctly, and so either the publisher must do the tagging in-house, changing the role of the copy editor, or software systems must be provided to assist the authors. One approach is to generate SGML-tagged files by taking documents produced by word processors such as MicroSoft WORD and WordPerfect and translating the internal coding into SGML tags. Another approach is to provide a 'smart' context sensitive editor that provides function keys for the principal document elements: these keys generate a visible effect on the screen and insert a tag in the file (e.g. the function key for a chapter might display the chapter heading centred on the screen and insert the `<h1>` tag in the file). Sophisticated editors might use the DTD to ensure that end tags are inserted where appropriate. (In practice, a combination of smart software and manual intervention by the publisher will probably be the norm.)

At the other end of the process it will be necessary to translate the SGML tags into something that can be recognised by the ultimate printing device, and until such time as there are SGML front ends for the popular formatters and composition systems used by commercial print shops the majority of publishers are unlikely to be interested in SGML-coded electronic manuscripts. These changes will take time — it depends not only on the development of technology but also on the publishers coming to terms with the new technology and changing their ways of working — so that although the benefits of standardisation will eventually be attained, we should not look for them in the short term.

A further point to consider is that much 'publishing' is either an in-house activity, with no need for document interchange, or within a community of users who share informal standards. For example, every IBM mainframe will have DCF/GML, LATEX is already a *de facto* standard in many parts of the academic community, and every Unix installation has troff with at least the ms macros, so document interchange is no problem. As an example, Unix comes with a machine-readable version of the Programmer's Manual, marked up with a particular set of macros designed for the purpose, which are also part of the standard distribution. So every Unix site can print out identical manual pages. At Southampton we use the same source files to display 'typeset' manual pages on the Sun workstation screen, and as a basis for a hypertext version of the manual. These benefits all derive from the use of a descriptive markup that predates SGML by many years. Systems like LATEX and troff/ms are mature and tested, whereas the public DTDs available for use with SGML at the present time are immature and untested. The established systems are also extensible, in the sense that it is possible to define new constructs as combinations of existing ones; the SGML user, by contrast, has to live with the DTD provided by the management, warts and all. Moreover, these existing systems are well supported, and come with an impressive range of author's tools, e.g. BIBTEX for maintaining bibliographies in LATEX documents, and *tbl* for processing tabular material in the Unix world, to mention but two. So although it is easy to translate SGML tagging into, say, LATEX form, many users will ask why they should bother with this intermediate stage? Why not use LATEX from the start?

The stock answer to this question is that the Data Type Definition defines not only the permissible tags, but their relationship, imposing a structure on the document, and that the SGML parser will check this structure. Whilst this is indeed the case, it is not a strong argument so far as a book or scientific paper is concerned. Such documents do not have a very complicated structure, and authors are unlikely to produce badly-structured documents, e.g. with the abstract after the first paragraph. In any case, systems like LATEX and the mm macros used with troff already do a measure of structure checking.

This is but one side of the argument, however. To appreciate the positive benefits to be gained from using SGML we should consider who actually uses it.

WHO USES SGML?

More precisely, we should ask who *really* uses SGML, since there are many who claim to use SGML when they are in fact just using a descriptive markup system to tag their documents. For example Robertson [35] describes the use of a generic markup system "based on SGML" under the title 'SGML Markup for Publishing at Leeds', and Crabtree [36], under the title 'Miles 33 and SGML' describes a system that uses a search-and-replace editor to translate arbitrary generic markup into the specific codes required by a commercial composition system. His reference to "the SGML command set" shows that he shares the common misconceptions about SGML. Similarly, we do not count use of IBM's DCF/GML system as using SGML[5]. By 'using SGML' we mean using the power of SGML to describe a complex document, and employing this description as an integral part of the processing of the document, to facilitate interchange between computer systems and/or to generate instances of the document using different media. This section does not aim to provide a comprehensive list of SGML users; it picks out a number of representative users to illustrate how SGML justifies itself.

The US Department of Defense — The ATOS project

Typical of the DoD work is the US Air Force Automated Technical Order System (ATOS) [37]. This is a system created to meet the needs of the publication, distribution and electronic delivery of Air Force maintenance manuals. The manuals are originated at one of five Air Logistic Centers (ALCs) and have to incorporate material received in electronic form from aerospace contractors. They have to be structured strictly in accordance with military specifications governing content and output appearance. They are produced first in draft form, then as final printed copy and also distributed in electronic form to the other ALC's. The use of SGML tagging with a specialised DTD enabled the objectives of the original requirement to be met, and has opened the way to constructing bibliographic search and retrieval systems. ATOS demonstrates a novel use of generic tagging. Each document must conform to limits set by the military specification regarding the reading grade level. Computing the reading grade involves a statistical analysis of the text (word lengths, sentence lengths etc.), and in performing this analysis some parts of the text have to be omitted. Since everything is tagged, it is easy to identify the text to be analysed, and if need be special tags can be incorporated for the purpose.

The Commission of the European Communities — FORMEX

FORMEX [38, 39] (Formalised Exchange of Electronic Documents) is a system developed by the Official Publications Office of the European Communities to facilitate production of the publications of the Commission, especially the official Journal of the Communities, and the archiving of these documents in electronic form. SGML has been chosen as the tool to support this mammoth task, with suitable DTDs being developed for the various documents.

[5] IBM has recently announced "SGML Translator: DCF edition" which incorporates an SGML parser. The reasons for this move and its implications are explored in the section 'The Future for SGML' below.

Her Majesty’s Stationery Office — legal text

Her Majesty’s Stationery Office (HMSO) is the UK Government printer and publisher. One of its responsibilities is the printing of the Statutes as passed by Parliament, and an ambitious project now under way is the creation of an electronic database of Statute Law. HMSO was one of the earliest users of computer-assisted typesetting, and developed its own typesetting system based on generic coding as long ago as 1972. From the beginning of 1987 statutes have been prepared using SGML coding with a complex (and evolving) DTD: the ultimate intention is that the same SGML source will drive both the typesetting system and the database archiving system, so that a new statute will appear simultaneously in printed and electronic form, but at present the text is first marked up with HMSO generic coding, and only after the printed version has been produced is this coding translated to SGML for the database archive. It is planned that this database will eventually contain all versions of a given piece of legislation, suitable cross-referencing being achieved by adding SGML tags. HMSO use of SGML is described by Stutely [40, 41] and Summers [42].

Hewlett-Packard — technical documentation

Documentation for Hewlett-Packard computers, software and instruments is produced at over fifty writing departments world-wide, and electronic interchange of material between departments is a major requirement. SGML is chosen as the means of facilitating this: an SGML parser called MARKUP [43], developed in-house, is used as a front-end to existing text-processing software.

Oxford University Press — the OED

The use of SGML in producing the electronic version of the Oxford English Dictionary (OED) has been mentioned already. The 21,000 pages of the current dictionary (12 volumes) and supplement (4 volumes) have been converted into a database of SGML-coded text. This text has been used in the first instance to produce a new printed edition of the dictionary in which the supplement is integrated into the main body, and a CD-ROM version of the original (1933) edition. The SGML database will form the basis for all future editions of the dictionary as new words are added, and also for variant editions of the dictionary.

Encyclopedia of Science and Technology — McGraw-Hill

The McGraw-Hill Encyclopedia of Science and Technology was prepared using SGML. This made it possible to produce it both in conventional printed form and as a CD-ROM: the textual material was then transferred to an on-line public database.

THE FUTURE FOR SGML

The applications described in the preceding section deal with documents of unusual complexity. In most of them electronic transmission of the documents is involved, and in many cases the document has to be produced in a non-traditional form (database, CD-ROM) in addition to the traditional printed form. SGML is very successful in meeting these requirements, and it might seem therefore that the future of SGML is as a niche product serving this specialised area. In fact, SGML will be much more widely used, and faces an assured future, since it has been chosen as a central pillar of the U.S Department of Defence CALS (Computer-aided Acquisition and Logistic Support) programme.

The CALS programme

CALS is a major initiative on the part of the US Department of Defense to control the mountains of paperwork associated with the design, development, manufacture, procurement, testing, deployment and maintenance of weapons systems. The scale of the problem can be appreciated by noting that the DoD spends more than 5 billion dollars each year maintaining and administering technical information on weapons systems. It stores more than 200 million engineering drawings, and the Navy alone deals with 200 thousand separate manuals. The objective of CALS is to produce an integrated system in which information is held electronically, and which interfaces to CAD/CAM systems, electronic publishing systems and databases both those within the DoD and those operated by the many defence contractors who supply the Department, so that it will be possible to receive, distribute and use technical information in digital form. To achieve this integration CALS is based firmly on international standards, and in particular includes SGML and ODA/ODIF as the prescribed standards for ASCII text processing. Defence contractors will be required to use SGML for their documentation, using DTDs supplied by the DoD as Military Specifications. For example, there is already a DTD for technical manuals that conform to Milspec MIL-M-38748B.

The effect of CALS

The immediate effect of the CALS programme has been to start a proliferation of SGML processors. Before CALS there were few SGML parsers available on the open market (though a number of companies had in-house implementations, e.g. Hewlett-Packard’s MARKUP processor described earlier). The readily available parsers were a system from Datalogics Inc. for the VAX range of computers and IBM PC (this was originally a by-product of the ATOS project), the MARK-IT parser from SOBEMAP for Unix systems and IBM PC’s (a development of the parser produced for the FORMEX project), and SoftQuad’s Author/Editor for the Apple Macintosh. The CALS initiative has caused manufacturers of main-stream technical documentation systems to adopt SGML. We have already noted the IBM DCF/GML development; Interleaf proposes to integrate the SOBEMAP parser into its Technical Publishing System TPS, one of the leading technical documentation systems; Scribe Systems (who might be said to have originated the descriptive markup concept) has integrated the SOBEMAP parser into a system called STEPS, and ArborText, originators of ‘The Publisher’ has announced SGML Writer, an interactive editor and word processor conforming to the SGML standard to run on Sun workstations under Unix and on IBM PC/AT class machines. Many other implementations will follow: the future of SGML seems assured.

SGML AND ODA/ODIF

Office Document Architecture (ODA) was originally developed by the European Computer Manufacturers’ Association (ECMA): it was taken over by ISO and together with the accompanying Office Document Interchange Format (ODIF) currently has the status of a draft International Standard (see reference [4]). ODA was designed as an interchange format for word-processor documents, and is intended for software-to-software communication rather than for direct use by a human user. ODA assigns two parallel structures to the text, a logical structure describing abstract relationships between components of the text, and a layout structure defining the positioning of elements on the printed page or display screen. Thus, for example, the logical structure might partition a document into chapters, sections and paragraphs, whilst the layout structure defines it in terms of pages, columns and margins. Both structures are simple hierarchies: at the lowest level in the hierarchy elements can be composed of text, line graphics, raster graphics etc., and ODIF describes the encoding formats to be used for each class of element.

There is clearly an overlap between SGML and ODA/ODIF and it has been claimed that SGML subsumes ODA/ODIF. This is not completely true, since SGML does not (yet) have anything to match the layout structure of ODA. The structures that can be described in SGML are much more complex than the simple hierarchies of ODA, and it is unlikely that one would want to use the power of SGML, with the concomitant load on the processor, in the context for which ODA was intended. It seems probable that ODA/ODIF will be used as a standard in the word-processing world, whilst SGML and its associated standards will retain their primacy in the realm of complex documents.

CONCLUSION

It is too early to come to a definitive conclusion about SGML. The availability of systems able to process SGML is not very wide, and in particular systems in which SGML is integrated with a formatter are only now appearing on the market. Moreover, SGML is designed as part of a suite of standards, and until the related standards are available and implemented we cannot finally pass judgement. (In this connection it should be noted that the traditional pace of development of international standards fits uncomfortably with the rapid advances of the technology. By the time the Standard Page Description Language is defined there will be so many PostScript engines in existence that the Standard will be largely irrelevant.)

It is clear that SGML has a valuable place in the preparation of complex documents, especially technical documentation, and has proved its worth both in regard to electronic interchange of documents and in multi-media dissemination of information. The take-up of SGML in the more traditional publishing areas of books and journals is less certain. Here the benefits of SGML do not so obviously outweigh those of the established and mature systems that are currently in use. A new generation of software products may change this: perhaps the most desirable result would be that by virtue of the markup minimization capability, together with smart editors, authors will be using SGML without knowing about it, whilst publishers reap the benefits.

ACKNOWLEDGEMENTS

I am indebted to Sebastian Rahtz for many illuminating discussions on SGML and for his comments on earlier drafts of this paper; also to the referees for their suggestions, which did much to improve the paper.

REFERENCES

1. *International Standard ISO 8879: Information processing—Text and Office systems—Standard Generalised Markup Language (SGML),* British Standards Institute, 1986.
2. Joan M. Smith and Robert Stutely, *SGML: The User's Guide to ISO 8879,* Ellis Horwood Ltd, Chichester, 1988.
3. Martin Bryan, *SGML: An Author's Guide,* Addison-Wesley, Wokingham, 1988.
4. *Draft International Standard DIS 8613: Information processing—Text and Office systems—Office Document Architecture and Interchange Format,* British Standards Institute, 1987.
5. C. Smith, 'Beyond Document Structure—SGML as a Software Development Tool', in *PROTEXT IV: Proceedings of the Fourth International Conference on Text Processing Systems*, ed. J.J.H Miller, Boole Press, Dublin, (1988).
6. J.H. Coombs, A.H. Renear, and S.J. DeRose, 'Markup systems and the future of scholarly publishing', *Communications of the ACM*, **30** (11), 933–947 (1987).
7. *The Chicago Manual of Style (13th edition),* University of Chicago Press, 1982.
8. J.F. Ossanna Jr., 'nroff/troff User's Manual', AT&T Bell Laboratories Computing Science Technical Report No. 54 (1977).
9. D.E. Knuth, *TEX and METAFONT: New Directions in Typesetting,* Digital Press, 1979.
10. —, 'Document Composition Facility: SCRIPT/VS Language Reference', SH35-0070-03, International Business Machines Corporation (1984).
11. The MIT Computation Center, *The Compatible Time Sharing System,* MIT Press, Cambridge, Mass., 1963.
12. J.H. Saltzer, *TYPSET and RUNOFF. Memorandum editor and type-out commands,* Project MAC memorandum (unpublished), 1964.
13. L. Lamport, *LATEX, A Document Preparation System,* Addison Wesley, Reading, Mass., 1986.
14. M.D. Spivak, *The Joy of TEX,* American Mathematical Society, Providence, Rhode Island, 1982.
15. B.K. Reid, 'A high-level approach to computer document formatting', in *Conference record of the Seventh Annual ACM Symposium on Principles of Programming Languages*, (January 1980).
16. B.K. Reid and J.H. Walker, *SCRIBE Introductory User's Manual,* Unilogic Ltd, Pittsburgh, 1980.
17. C. Goldfarb, 'A generalised approach to document markup', *SIGPLAN Notices*, **16** (6), 68–73 (1981).
18. —, 'Document Composition Facility Generalised Markup Language Implementation Guide', SH35-0050-2, International Business Machines Corporation (1985).
19. Joan M. Smith, 'The Computer and Publishing: An Opportunity for New Methodology', in *PROTEXT II (Proceedings of the Second International Conference on Text processing Systems,* ed. J.J.H Miller, Boole Press, Dublin, pp. 107–113, (1985).
20. J-M. de la Beaujardiere, 'Well-Established Document Interchange Formats', in *Document Manipulation and Typography*, ed. J.C van Vliet, Cambridge University Press, Cambridge, pp. 83–94, (1988).
21. J.R.R. Tolkien, *The Lord of the Rings,* George Allen & Unwin, London, 1954.
22. D.J. Becker, 'Standards: The Interstructure for Electronic Publishing', *The Seybold Report on Publishing Systems*, **18** (2), (1988).
23. Joan M. Smith, 'The Standard Generalised Markup Language (SGML): guidelines for editors and publishers', British National Bibliography Research Fund Report 26, British Library (1987). ISSN 0264-2972; 26
24. Joan M. Smith, 'The Standard Generalised Markup Language (SGML): guidelines for authors', British National Bibliography Research Fund Report 27, British Library (1987). ISSN 0264-2972; 27
25. —, 'Standards for Electronic Manuscript Preparation and Markup', in *Electronic Manuscript Series,* Association of American Publishers, Washington D.C, (1986).
26. S.P.Q. Rahtz, 'Bibliographic Tools', *Literary and Linguistic Computing*, **2** (4), 231–241 (1987).
27. —, *Electronic Manuscript Project; The Markup of Tabular Material,* Association of American Publishers, Washington D.C, 1986.

28. M.E. Lesk, ‘tbl — A Program to Format Tables’, AT&T Bell Laboratories Computing Science Technical Report No. 49 (1976).
29. —, *Electronic Manuscript Project; The Markup of Mathematical Formulas,* Association of American Publishers, Washington D.C, 1986.
30. B.W. Kernighan and L.L. Cherry, ‘A System for Typesetting Mathematics’, *Communications of the ACM*, **18** (3), 151–157 (1975).
31. *International Standard ISO 8632: Information processing systems—Computer Graphics—Metafile for the Storage and Transfer of Picture Description Information,* British Standards Institute, 1986.
32. E. Weiner, ‘The electronic English dictionary’, *Oxford Magazine* 6–9 (1987).
33. T. Benbow, ‘The New Oxford English Dictionary Project: an Introduction’, *SGML Users’ Group Bulletin*, **1** (2), (1986).
34. D. R. Raymond and F.W. Tompa, ‘Hypertext and the Oxford English Dictionary’, *Communications of the ACM*, **31** (7), 871–879 (1988).
35. L. Robertson, ‘SGML Markup for Publishing at Leeds’, *SGML User’s Group Bulletin*, **3** (1), 20–21 (1988).
36. M. Crabtree, ‘Miles 33 and SGML’, *SGML User’s Group Bulletin*, **3** (1), 22–24 (1988).
37. Pamela L. Genussa, ‘Document Preparation Method of the United States Air Force Automated Technical Order System (ATOS)’, *SGML Users’ Group Bulletin*, **2** (1), (1987).
38. —, *FORMEX: Formalised Exchange of Electronic Publications,* Office des Publications Officielles des Communautés Européennes, Luxembourg, 1985.
39. C. Guittet, ‘FORMEX: une Mise en Practique des Normes Internationales’, *SGML Users’ Group Bulletin*, **1** (2), (1986).
40. R. Stutely, ‘HMSO and Generic Coding — Past, Present, Future’, *SGML Users’ Group Bulletin*, **1** (2), 61–64 (1986).
41. R. Stutely, ‘HMSO’s Database of Legal Text’, *SGML Users’ Group Bulletin*, **2** (2), 91–93 (1987).
42. I. Summers, ‘The Application of SGML to Statutory Instruments’, *SGML Users’ Group Bulletin*, **2** (2), 94–96 (1987).
43. Lynne A. Price, ‘MARKUP: Hewlett-Packard’s SGML Implementation’, *SGML Users’ Group Bulletin*, **2** (2), 116–118 (1987).