Markup Reconsidered

Darrell R. Raymond
Frank Wm. Tompa

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

Derick Wood

Department of Computer Science
University of Western Ontario
London, Ontario, Canada
N6A 5B7

ABSTRACT

We describe some of the implications of markup for document management systems. Markup's properties are inherited from text, since it is embedded in text. These properties are most advantageous when document structure is reducible to substrings of characters, and when the update characteristics of the structure are similar to the update characteristics of the text. We describe situations in which these characteristics are disadvantageous. Markup is not a data model, but one of several possible techniques for representing structure. For this reason it should not be the foundation of document management systems.

20 March 1995

This paper was presented at the First International Workshop on Principles of Document Processing, Washington DC, October 21-23, 1992.

Markup Reconsidered

Darrell R. Raymond
Frank Wm. Tompa

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
N2L 3G1

Derick Wood

Department of Computer Science
University of Western Ontario
London, Ontario, Canada
N6A 5B7

1. Introduction.

A sound theory of document management systems requires a good understanding of the role of markup. Markup is the use of embedded codes, known as tags, to describe a document's structure, or to embed instructions that can be used by a layout processor or other document management tools. Markup is ubiquitous, but its properties as a form of data representation have not been studied by the computer science community. Markup has received attention largely in the context of document standards or implementations.

There are several reasons to consider markup as an independent phenomenon. The first is that it is pervasive; wherever text is used as a type of data, markup is likely to be employed. This usage includes not only traditional online and offline documents, but also programming-language source code, command languages, macro languages, text-based protocols, and other sequential information. The second reason to study markup is that our current understanding of it is bound to the operations of markup processors; that is, we usually explain a tag's function by describing the operation of the processor for that tag. We know that in other areas of computer science, operational descriptions of phenomena are not sufficient[8]. As a result, many existing document systems and standards are idiosyncratic and application-dependent, and do not appear to be founded on sufficiently general principles[18]. A third reason is that a better understanding of markup might help us to clarify exactly how document systems differ from traditional databases. Text appears to be quite different from numbers, and markup seems an unlikely candidate for structuring accounting databases, but these are observations of established practice, not of theoretical limits. If text is different from traditional data, it would be useful to

This paper was presented at the First International Workshop on Principles of Document Processing, Washington DC, October 21-23, 1992.

find some fundamental reasons for that difference, and a better understanding of markup may help us to identify them.

In what follows, we develop some of the aspects of a theory of markup, independently of particular implementations or standards. We begin by characterizing text's properties as a system of symbols. Since markup is embedded in text, it inherits many of text's properties. Next we consider how some of markup's problems are a result of these inherited properties, and hence not easily avoided. Finally, we briefly explore how document management systems differ from traditional databases, and consider how markup addresses these unique needs.

2. What is markup?

What we think of as text is irretrievably bound to the processes used to create it and to access it. Various types of markup have been invented to support these processes. Historically, the most important processor of documents has been the human reader. Thus, the earliest types of markup were designed to facilitate the reading process. In consonantal scripts, for example, diacritics were employed to signify vocalic distinctions[9]. Punctuation, word division, and sentence division are a type of markup adopted to provide the reader with clues about emphasis, organization, and breathing. These factors may seem secondary to the content of the text, but note that:

> It probably also makes a difference whether in Joyce's *Ulysses* one of Stephen Dedalus' first thoughts is 'No mother' (as in printed versions) or 'No, mother' (as in the manuscript)[32].

Punctuation symbols are in such common use that they seem themselves part of the alphabet, and so seem part of the content of the text. Another early type of markup to aid reading is illumination, the graphical embellishment of important characters in medieval manuscripts. Illumination serves a decorative purpose, but it also identifies important breaks in the text, such as new chapters and sections. Page furniture—headers, footers, and page numbers—is another type of markup that helps to orient the reader.

Like speech, documents purport to describe the world or tell a story. Unlike speech, however, documents are considered objects in their own right, separate from the speaker, and so are often themselves a subject for documentation[13]. This property of documents led to a second class of markup, namely critical commentaries of the 'main' content. This class of markup includes marginalia, footnotes, and other explanatory meta-content. Scholarly documents make extensive use of critical commentaries. Some editions of ancient works are so heavily marked up that the main content is only a small fraction of the total information.

The advent of book making and the consequent specialization of the writing, editing, and book-production crafts led to a third class of markup: editor's and printer's marks. This class of markup supports 'out-of-band' communication between writer, editor, and bookmaker, and can be thought of as a kind of critical commentary on the document's production, to be read at the next stage of the production process. Production markup is typically applied directly to manuscripts or proofs, usually with coloured pens and a set of symbols agreed upon by convention.

The first generation of online typesetting systems adopted embedded markup to communicate layout instructions from the human to the machine. Since the primitive input devices of the time did not support colour or arbitrary glyphs, markup and content was entered in the same input stream, using the same alphabet, and markup was distinguished by special prefixes and suffixes. Most computer typesetting systems still use embedded commands to control document production.

Currently markup's evolution is driven by two basic requirements. First, the requirement for multiple presentations of a text has resulted in markup becoming an indirect specification of presentation. Rather than denoting specific layout instructions for document elements, markup of design elements identifies members of specific layout classes that are mapped to a specific presentation through a style sheet. Markup of design elements (such as the *-ms* macros for *troff* and LaTeX macros for TeX) facilitate layout parameterization. Second, the idea that documents can be treated as databases has resulted in markup assuming a role as representation for data elements, facilitating automatic querying and updating. Markup thus plays an important role in identifying structure.

In modern use, there are several recognized subcategories of markup. *Descriptive* markup identifies a logical structure in a document[7]. Descriptive markup is usually distinguished from *procedural* markup, in which tags are mapped to actions of a specific device. Markup that is not restricted to a single application, style, or formatting system is known as *generalized* markup[10]. Generalized and descriptive markup are often taken to be synonymous. Recently, markup advocates have suggested a useful separation between procedural and *declarative* markup, where the former describes an operation and the latter describes a constraint. They also separate *presentational* and *analytic* markup; the former is markup that describes appearance, and the latter is markup that describes an ontology for the document.

What commonalities exist between punctuation, critical commentaries, printer's marks, typesetting commands, and structural tags? The essential characteristic of all these forms of structure is that they are simultaneously *embedded* and *separable*; they are part of the text, yet distinguishable from it.

Structure can be either strongly or weakly embedded. In either case, the structure is present in the data; the difference is that for strongly embedded structure, its position in the data is information bearing. An example of strongly embedded tags is found in the following text:

```
now is the time for all good <sexist>men</sexist>
to come to the aid of the party
```

These tags demarcate a specific word in the text and assign that word to a category. An example of a weakly embedded tag is the following:

```
now is the time <sample id=2> for all good men to
come to the aid of the party
```

The tag `<sample id=2>` is informative, but its location within the text is not information bearing; in this example, it could be placed at any point in the text, or even outside it, without losing its meaning. The test for strongly embedded markup is: Can the tag be moved to some other position in the text without loss of information? If it cannot, then the tag is strongly embedded.

In normal use, 'markup' means strongly embedded structure, and the remainder of our discussion focuses on the properties of this kind of markup. Definitions are always debatable, but we will treat this one as both necessary and sufficient; anything that is not embedded and separable is not markup, and being embedded and separable is sufficient for something to be markup. Note, therefore, that we do not require a structure to be descriptive, or context-free, or nested, or distinguishable by means of unique start and end characters, in order to be markup. These are all properties of specific forms of markup, rather than characteristic of markup in general.

A word should be said about so-called *out-of-line* markup, non-embedded structure that conforms to the syntactic requirements of a given markup standard[2]. Out-of-line markup is not markup by our definition, but readers can choose the terminology they prefer; we intend merely to distinguish between internal and external structure. While out-of-line markup does not suffer from some of the disadvantages that we shall discuss, neither does it share the advantages of strongly embedded markup, and is more properly considered a specific type of external structure.

In addition to describing what markup is, it is important to note what markup is not. It is not a computational automaton, a data model, or a mathematical formalism. It is not a computational automaton, because it does not by itself specify the input, the output, or the behaviour of any abstract device. It is not a data model, because it does not by itself denote classes of data, operations on those classes, or constraints on valid instances. Finally, it is not a mathematical formalism, because it is not accompanied by a set of rules that describes how it can be manipulated. Markup is, instead, simply the denotation of specific positions in a text with some assigned tokens. It is fully dependent on external information for meaning. In most document management systems that employ markup, a computational device (usually a pushdown automaton) is implemented as a way of embedding a mathematical formalism (usually a context-free grammar) in a system that recognizes texts containing structure (usually indicated by the presence of markup). Clearly, it is possible to have the formalism (and the implied data model) without either the computational device or the markup. Markup thus belongs not to the world of formalisms, but to the world of representations; its properties of interest are those of any data structure.

3. The properties of text.

Markup's properties are largely derivative of the properties of the documents in which it is embedded. Thus, it is worthwhile to review some of the basic properties of documents. In the following, we restrict ourselves to the properties of text, particularly to texts as they are usually represented in computers. It would be interesting to develop a theory of documents that would include such things as paintings and gestures, but we will not attempt to do so here.

The class of text is an important subclass of the general family of sign systems. Texts are constructed from atomic elements known as characters. A set of such characters constitutes an alphabet. Alphabets partition the possible set of inscriptions; an inscription can belong to at most one character in an alphabet. Moreover, it is the case that the level of discrimination needed to make such distinctions is finite[11]. The number of equivalence classes defined by an alphabet is always finite, but can vary in size. The Roman alphabet defines 26 characters, whereas the Chinese alphabet defines 50,000 or more.

Individual characters by themselves are rarely sufficient communications. An essential feature of a system of writing is that it supports the creation of compound expressions by means of character combinations[12]. Most texts use relative positioning to produce these combinations; characters are placed next to one another, and a collection of characters that share proximity is treated as a unit. Exceptions to this principle are situations in which multiple characters occupy the same position; for example, the use of diacritics for accenting, or the use of overstrikes in artificial languages such as APL.†

Symbol systems that employ discrete characters and relative ordering have at least four classes of properties: resources, granularity, order, and update. These four classes are somewhat interdependent, but we present them separately for the purposes of exposition.

Resources. The resources of a text are the raw materials of which it is constructed: its symbol set and the space in which the symbol set is laid out. Resources are important because they are scarce; there are neither an infinite number of symbols nor is there an infinite amount of space. In order to create new words, one must use some combination of existing symbols that requires space. Resources are also important because installing them and reading them requires effort. Accordingly, in many languages the most frequently used words are short.

The scarcity of resources affects electronic markup in the same way that it affects the main text; it encourages brevity. Where markup is added manually, there is a strong desire to keep the most frequent forms of markup concise, so as to reduce input effort. Even a verbose markup scheme such as LaTeX has commendably brief markup for mathematics and for some frequently used codes, such as those for paragraph breaks. Markup minimization techniques in SGML are designed to reduce the space consumed by markup, and the effort involved in installing it. Where markup is added automatically, there is less need to worry about input effort, but even here it is undesirable to be profligate with markup, since it interferes with the readability of the underlying text.

Granularity. Granularity is a measure of the power to distinguish detail, or the level of atomicity of the system. The granularity of markup is no greater than that of the text in which it is embedded; tags cannot demarcate more precisely than a single character. This is important if one considers the alphabet to be a discrete sampling of a continuous medium. Phonetic alphabets are samplings of at least two continuous media—speech and ideas. As in any sampling process, an upper bound can be given for the information in the sampling, based on the frequency of the samples. This bound may be further degraded if the samples are not well chosen. For example, if the Roman alphabet were accurately mapped to phonemes, there would be less need for the International Phonetic

†Alternatively, it is possible to regard overstrikes as an extension of the alphabet.

Alphabet. We would have alphabetic distinctions (rather than contextual ones) for *sow* (female pig) and *sow* (to plant), and there would be no need for ligatures to represent certain vowel combinations, as in *hæmorrage* or *pæleozoic*. The potential for mismatch between words and ideas is evident in the difference in precision of words used in the social sciences compared to those used in the natural sciences[17].

Granularity affects markup in two ways. Markup's ability to distinguish structure is ultimately limited by the level of distinction that the text itself provides; tags can do no more than define superclasses of the atomic elements already in the text. Second, markup is itself text, and so itself exhibits granularity. The number of tags and the rules for their combination also constitute a fundamental limit on the expressive power of markup.

Order. Order in text, as we have observed, serves to redress the limitation of a small alphabet by permitting position-dependent combination of characters. Text is generally ordered linearly, as a single stream of characters. This order is used because for most of the history of texts, the processor has been a human being, reproducing the text as speech. The reproduction order of text is a total order; every element in the text is ordered with respect to every other. Moreover, this order is achieved in a relative, rather than an absolute fashion; that is, every atomic element is related to just two others (its predecessor and its successor). Documents that are not primarily intended to be rendered into speech often contain non-linear elements, such as tables, marginalia, cross-references, aligned translations, and footnotes.

In electronic uses, order typically serves as the controlling sequence of markup access; the markup processor encounters the markup in the order in which it occurs when the text is consulted in reproduction order. Thus, typesetting markup is usually found next to the document element whose presentation it modifies. Markup systems can also rely on the total order to indicate substructure in the markup; for example, paired tags that are contained within some other tag pair are usually considered to be nested within the latter pair. In this case, the total order is viewed as a depth-first traversal of the implied hierarchy.

Update. The properties discussed so far have been static ones. Electronic text also has specific dynamic properties. There are two important update characteristics of text, derived from the order of the text and its redundancy. Order in the text is specified in a relative fashion; this means that the insertion of characters can be done by breaking only the relationship between the two adjacent characters that surround the insertion point; all other characters simply maintain their relative position.† The inserted fragment now participates in an ordering relationship with all other characters in the text, and does so without the need for transitive specification of these relationships.

The second update characteristic is due to the inherent redundancy of text. In order to update a given word (for instance, to change its spelling), it is necessary to locate all copies of the word and change them each individually. The standard representation for text, then, maintains position information as concisely as possible, but keeps many copies of any given word.

†While not literally true in most text editor implementations, this is the illusion they support. The actual implementation of insertion often requires that the positions of all of the succeeding characters are actually updated, because they are stored in absolute order rather than relative order.

Markup inherits its update characteristics from text. The insertion of a tag into a text is a relative update that attaches to the tag all the dependencies that are implied by that location (for example, it may become a member of all the tag pairs that surround it), without the need to specify them explicitly. The property of redundancy is also present; in order to change a tag's name it is necessary to locate all copies of the tag and change them individually. Redundancy makes converting presentational markup to descriptive markup difficult, since each instance of presentation must be separately considered.

The four classes of properties interact in roughly the following manner: A desired level of granularity determines a set of atomic expressions, known as an alphabet, and also determines the rules for creating compound expressions. The alphabet, the rules, and the medium are collectively the resources at the disposal of an author. These resources are used in creating a text. The text is ordered and redundant, and therefore has specific update characteristics.

4. Implications for markup.

The properties of text bear on markup's design and use. These properties are also at the root of the more widely known problems with markup. We now consider some of these problems in more detail.

As mentioned earlier, the resources of a text are its symbols and its positions. Markup, as embedded text, consumes both symbols and positions. Consider the consumption of symbols. In order to be part of a text, markup must use textual symbols. In order to be separable from the text, something must flag the distinction between markup and content. One approach to separability is to partition the character set, choosing markup characters from one subset and content characters from another. Partitioning is widely used to separate punctuation characters from alphabetic characters. Partitioning is also used in some popular word processors that keep 'hidden' bytes, chosen from outside the ASCII character set, to store formatting information. We might call partitioning *logographic* markup, since it resembles logographic writing systems in allocating a unique symbol to each 'word'. In both cases the disadvantage is clear; logographic systems require large, and possibly unbounded, alphabets.

The more usual approach to separability is to use the same character set for both markup and content, but to isolate markup and content in separate streams or modes. Modes can be identified in several ways; for example, one might maintain external structures that indicate which parts of the text are content and which are markup. The more common approach, however, is to allocate a small number of characters (or character sequences) to indicate a mode switch. This approach is adopted because it is supportable by parsers. Dedicating even one character to mode switching is irritating, however, since it seems inevitable that such characters need to appear as content occasionally. Thus, an escape technique must be used (typically, doubling the mode switch character or using another character as an escape indicator) to flag that the mode switch character should be interpreted as a literal. Escape sequences must themselves be escapable, and so on. Though no formal difficulties are raised by this technique, few systems implement it flawlessly. UNIX users, for example, are familiar with the dilemma of trying to determine how many quotes and backslashes are needed to protect a command line argument from shell interpretation. Another example is the C++ template facility. A template parameter list is bounded by angle brackets, and no spaces are necessary in constructs like:

```
template <class T> class set;
set<int> a_set;
```

If the parameter list itself contains a template, however, then a space *is* required to help the system distinguish the end brackets:

```
set<set<int> >   a_set_of_sets;
```

A third example might be SGML's record separator problem.

The resource of positions can also result in problems. If the text does not permit overstrikes, then there can be at most one character at each position. Positions thus become a scarce commodity, especially for situations in which more than one tag ought to occupy a single position. In a descriptive markup system, for example, it is usual to demarcate the start and end of a text fragment with tags that correspond to its element type. If a text fragment belongs to more than one element type, however, one is faced with the problem of putting two or more tags at the same position in the text. Some possible taggings for such a situation include:

(i) <X><Y>--------</Y></X>
(ii) <X><Y>--------</X></Y>
(iii) <X Y>--------</X Y>

The first solution identifies the fragment as both an X and a Y, but cannot be distinguished from a nesting of the two elements. The second solution is distinguishable from nesting, but is not context-free. Neither of the first two solutions is robust to insertions between <X> and <Y>, since they will change the relationship from coincidence to overlap. The third solution combines the two elements, is distinguishable from nesting, and is robust to the problem of insertions. However, it requires a local overriding of the order of the text, since the order of the identifiers X and Y must be immaterial. In effect, this defines an equivalence class consisting of the tags <X Y> and <Y X>. The third solution is thus a form of logographic markup, since we are extending the alphabet to include new equivalence classes of characters.

Consumption of positions has other deleterious effects. Embedded tags can affect any search based on relative position, such as proximity or regular expression search. Tags can inflate the distance between two text fragments so that they fall outside the bounds specified for a proximity search; if tags appear within words (consider markup of diacritics or ligatures, as in representing ü by `ü`) they can defeat regular expression searching that does not take the markup into account, so that a search for `f.r` matches *for*, *fir*, *far* and *fur*, but not *für*. The insertion of tags also affects document processing systems that rely on absolute positions. One consequence is that documents that are marked up 'on the fly' must keep track of the original positions of the text in order to provide useful error messages[16]. Some documents cannot be modified by inserting markup because their absolute positions are indexed by other tools in the environment. An important example is program source code, in which characters' positions may be known by debuggers, syntax-directed editors, code profilers, compilers, and source code control systems.

Let us now look at the implications of granularity for markup. Because it shares the data's representation, markup is bound to the granularity of the representation: it shares the data's level of discreteness. The chief effect of granularity is to make it difficult to express structure that is not a subset of character positions in the text. Thematic structures are one example; Hamlet may have moments of indecisiveness, but these are not well correlated with unique character sequences in the text of the play[28]. Thematic structure, like many categories, is defined not by a set of necessary and sufficient properties, but instead by possessing a sufficient number of characteristics, none of which are essential[20, 3]. Probabilistic or statistical information about the text is also unlikely to have discrete boundaries. The position of marginalia and other non-sequential commentary is not precise to within a single character. High level structures are not always compositions of low level features; people have two arms, but this is not a consequence of their possession of two kinds of cells, those that belong to arms and those that do not (consider blood, nerves, ligaments). In general, structure is not always reducible to a functional description of a system's subcomponents[23].

Granularity problems often show up as indeterminacy in applying descriptive markup to white space and punctuation. As these elements are themselves markup (though for a different function), it is not surprising that there are problems in applying another level of markup to them. Sentences that end in full-stop abbreviations, for example, will not have an extra full stop to signal the end of the sentence. If markup is used to identify the abbreviation, is the punctuation part of the abbreviation or part of the sentence? If punctuation or white space is included within the entity, it may be erroneously removed if the entity is modified. White space, when used as a delimiter for both markup and text, can become confused if markup is treated as optional (as in the case of SGML's CONCUR). The interaction between punctuation and markup can also complicate string searching.

A third class of difficulty arises from markup's inheritance of the text's order. Text is totally ordered; thus markup's most natural use is to express orders that are consistent with this total order, such as hierarchies. It is interesting to speculate whether the widely accepted view that documents are hierarchical is a result of deep thought about document structure, or simply a result of years of experience with marked up texts. Whichever explanation is true, it has also been said that the interesting structures in documents are largely the non-hierarchical ones[24]. Proposed future applications such as hypertext and active documents are also non-hierarchical. Even the table, however, produces problems for markup and other systems that are based on a total order. Consider the following table:

	A	B	C
X	24	55	19
Y	3	0	123
Z	1	1	1

Instances of tables can be captured as sequentially tagged texts. For example, the preceding table might be represented in the following way:

```
center;
c | c c c.
    A   B   C
_
X   24  55  19
Y   3   0   123
Z   1   1   1
```

The first problem with representing tables as marked-up texts is the scarce resource of positions, described previously. The words in a text belong to a single sequential relationship, but the 'words' or data values in a table belong to multiple orthogonal relationships. Thus, tables exhibit the need to identify a text fragment as an instance of more than one element type. In the table preceding, the value 19 is both a C and an X, but only one of these facts is 'near' the value.

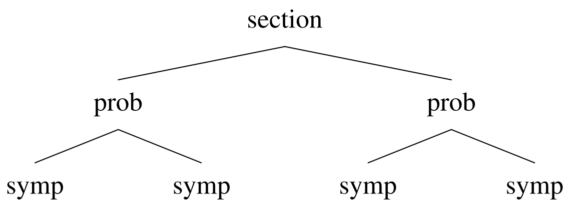
The second problem is overspecification. We might conclude from the marked-up representation of the table that 123 'follows' 19, since it comes later in the sequence. But the table does not require this, unless it is the case that Y follows X. The linear structure of the text is an overspecification of this (and any other) set-valued structure.

A third interesting difference between tables and sequential text is the difference in normalization. As mentioned earlier, texts are highly redundant in their data values, but not redundant in their use of positions. Tables, on the other hand, attempt to show each datum only once, adjusting the layout so that the relevant dimensions impinge on the data values in as natural a fashion as possible.

The normalization implicit in tables and the redundancy implicit in texts is evident in the different update operations needed for each. To update a text, we typically use an editor with a search-and-replace command, performing a redundant update to each copy of a data value. To change the word 'Constantinople' to 'Istanbul', we perform a redundant operation on every instance of the word. To update a table value, on the other hand, we simply find the single copy of the value and change it, and the change is automatically 'available' to each of the relevant dimensions. In the preceding table representation, we need change the value 19 only once to affect both C and X.

Structural update of tables represented in linear form is a different matter, however. The preceding marked-up table supports row interchanges, since this can be done simply by interchanging lines of the text file, but does not support column interchanges, since this requires editing each line separately. Transposition of the table is another operation that would be exceedingly tedious in this representation. Table editing is generally not well supported in most environments[31], and it is not our purpose here to complain that markup does not do better. Rather, we wish to emphasize that the order of text inherited by markup is the fundamental reason for this problem.

The overspecification of order can also be a problem even for apparently benign nested texts. Consider the following fragment from a hypothetical repair manual that lists problems and their symptoms:



In marked-up form, this might appear as:

<section>...<prob>...<symp>...</symp>...</prob>

The symptoms are ordered within the problems, and the problems are ordered within the sections, but the order of the symptoms within the section has no meaning. The total order of the text is an overspecification of the partial order involved in the structure. As was the case with granularity, structure is not always reducible to character positions.

The representation of non-linear orders is also a problem. Markup advocates have given considerable thought to the problems of representing multiple and discontiguous structures. These problems have never been seen to be fatal; on the other hand, a resoundingly satisfactory solution to such problems has yet to be devised[1, 2, 21, 14, 22].

The fourth and final class of difficulties arises from markup's inheritance of the update properties of the text. The virtue of sharing both update properties and resources is that structure can be updated with exactly the same tools that are used to update the text itself, thus obviating the need to develop a tool specifically designed for structure update. In batch typesetter environments, for example, users commonly employ their standard text editors to input the text and add the markup commands to that text.

Updating markup as if it were text is acceptable when the update characteristics of the markup and the text are comparable. In many situations, however, the update characteristics differ. Some structures are much more dynamic than the text; consider for example identifying the structures 'the word I am currently reading', or, in program source code, 'all uses of the variable X after the nth iteration of loop Y' (where n is specified dynamically)[25]. If these structures are to be represented by markup, then we need to insert and delete tags dynamically while using the text. Dynamic structure thus requires that the text be updatable; if the text is read-only, then so is its structure. Markup inherits the update properties of the text, but in general the update properties of structure and content should be independent of one another.

A second update problem is that as markup has become more complex, it is less likely that text editing tools will be sufficient for editing markup. When markup consisted mostly of simple strings that were largely context-independent (as in batch typesetters), the use of a text editor to update markup was reasonably reliable. Some markup systems, on the other hand, permit multiple attributes, insist on tag pairing, even though members of the pair are widely separated (although sometimes end tags can be eliminated), allow ID values and cross-references to other tags or positions in the text, and usually require conformance with some external structure such as a grammar. Complex markup, then, has reduced the original update advantage; many SGML texts should not be edited with simple text editors, and retaining this capability simply risks well-intentioned but erroneous manual editing.

Markup exhibits many properties that are not shared by non-embedded structures. Non-embedded structures need not consume text resources, need not limit themselves to the granularity of the text, need not inherit the order of the text, and need not inherit the update characteristics of the text.

5. Alternative representations for text.

Our investigation of the implications of text characteristics for markup has presupposed the standard electronic representation: a list of characters ordered in accordance with their appearance in an output medium. This representation seems so natural that its status as the definitive form of 'text' is not even questioned. There are other representations for text, however, and by considering some of them we expose other issues in the design of document management systems.

Consider the following representations, each intended to be a marked-up version of the same text:

(i) <sol>to be or not to be; that is the question...</sol>

(ii) <sol><6><2>be</2></6><8>is</8><4>not</4>
<3>or</3><10>question</10><7>that</7>
<9>the</9><5><1>to</1></5></sol>

(iii) <sol>&tb; or not &tb;; that is the question</sol>

Representation (i) is a straightforward sequential text string with embedded markup. Representation (ii) is a sequence of words, ordered alphabetically, and marked up in such a way that the text can be reproduced by following the sequence defined by the tag labels. Representation (iii) is again a sequential text, but some phrases have been factored out and are represented as external entities; the tag `&tb;` maps to the phrase 'to be'.

The choice between these representations is usually made on operational grounds; one chooses based on estimates of the need for compression, for particular types of expected query, or for control of update anomalies. Representation (i) requires far fewer tags and is easily read; however, it contains multiple occurrences of some words and so may exhibit inconsistent spelling. Representation (ii) appears to be less redundant, storing only one copy of each word. It can easily be used to generate a word list and frequency counts, but requires more processing to reproduce the canonical form of the text (of course, technique (i) requires more processing to produce the word list and frequency counts). Representation (iii) is a middle-ground solution, with some redundancy eliminated, and only a medium amount of effort required to read the text (much of the text is already in canonical order, but there are frequent diversions to access external information).

Representation (ii) may seem far-fetched to those for whom it is patently obvious that texts are ordered sequences. But this technique is simply a tagged version of structures that we otherwise recognize as the common concordance, the inverted list, or the back-of-the-book index. Each is an organization of a text to support a specific kind of access. Each is a complete structure, containing the whole text; otherwise it would not be possible to effect reconstruction of the Dead Sea Scrolls from a published concordance[30].

Operational concerns are important. However, they tend to strengthen the argument that issues of markup are issues of representation of structure, and not of its abstract form. There is one abstract issue that is implicit in the choice of representations, common to many data modelling situations: deciding what constitutes ‘one thing’[15]. The choice of representation can be an implicit statement of our notion of object, because each structure is optimized to manage a specific type of object. Representation (i) is designed to store not characters, but the positions that they occupy. Thus, it distinguishes between and maintains the identity of positions. Representation (ii) is designed to store individual words. Since the sequence of the ‘original’ text is captured in the order implicit in the tag identifiers, the sequence in the data structure is free to be used for some other purpose—in this case, the lexicographic order of the words. Representation (iii) stores the positions occupied by a mixed bag of characters and phrases, and is designed to maintain the identity of both the positions and the phrases.

Issues of identity are subtle, but not without impact. In a relational database, two tuples with the same key in fact denote only one thing. In a traditional text database using representation (i), two occurrences of ‘to be’ are still separate objects; in representation (iii) they are copies of the same object; in representation (ii) they are made up of copies of the objects ‘to’ and ‘be’.

Identity is also important in normalization, an issue that has remained largely hidden in document management systems. Normalization is the process of adjusting the design of a database so that updates will not result in anomalies. One of the benefits of normalization is reduced redundancy. The three representations differ in the types of redundancy they permit. Representation (i) has redundancy in words (hence, as noted earlier, the potential for inconsistent spelling), whereas representation (ii) has removed the redundancy that facilitates this inconsistency. Representation (i) has eliminated redundancy in position, whereas representation (ii) may have redundancy in positions; that is, it permits many instances of tag pairs <n>...</n>, for some number n, thus allowing synonyms (e.g., <65>Constantinople</65>...<65>Istanbul</65>) to occur in the text. Representation (iii) has removed the redundancy in some common phrases, but it has also removed the redundancy of position.

Elsewhere we have argued that software design is often concerned with the choice between normalization and automatic processing. Systems can be constructed based on normalized data, or they may track existing redundancy and perform redundant operations as necessary when updates are requested[27]. Similarly, in document management systems, we often choose between redundant storage of information and automatic processing. Such issues can be described in formal terms for traditional databases, but we are not yet capable of doing this for document databases.

In traditional database management systems, normalization is based on explicit statements of data dependencies, such as functional and multivalued dependencies. In traditional document management systems, such dependencies are implicit in the operational descriptions of the processors. Representation (iii), for example, defines an implicit dependency between the external phrase and the entity reference placed in the text. The implicit use of dependencies is not necessarily bad. However, we must at some point distinguish between the formal aspects of the dependency structure and their operational effects. The former belongs to a theory of document management systems, while the latter belongs to discussions of their implementation.

6. Requirements for document management systems.

Document management systems are a type of database management system, and thus will share with traditional databases issues such as storage, searching, normalization, update through transaction control, the possibility of distributed data and the possibility for parallel processing. But document management systems are not simply reducible to existing database systems, because documents have specific requirements that are not common in traditional database processing. We next explore some of these requirements, and consider markup's ability to address them.

Modelling. The first area of difference between text databases and traditional databases is the likelihood of schema update. In traditional databases, the data model is usually static; it is carefully determined in advance, as a definitive statement of the organization or world it is intended to represent. When the data is loaded, modelling is effectively terminated; the data will change, but the model of the world generally stays constant or evolves slowly. In document databases, on the other hand, putting text online is the first, not the last, step in developing a data model. Where numeric data is an interpretation of the world, many texts are themselves worlds to be interpreted, and thus the need for modelling is potentially infinite. Each model of a text can point to different structures, different elements, and different constraints, since an essential property of interpretation is creativity. Thus, in document databases, there is often a need for dynamic modelling.

Markup's ability to fulfill this need is severely limited. The binding of markup's update properties to those of the text stifles the addition of new data models. Markup's use of text resources means that such additions (and hence, changes to the text) will have an effect on both internal structure (existing markup) and external structure (pointers into the text). Markup's granularity, as we noted earlier, also limits its ability to represent structures that are continuous in some way. Complaints about markup standards are sometimes indicative of the fear that markup will interfere with the free interpretation of texts[21].

Independence. A second area of difference between text and traditional data is in their differing views of how to achieve modularity and independence in the systems. The document community focuses on ensuring modularity at the level of the data, by insisting on the interchangeability of documents. Given an appropriately marked-up document, it should be possible to transmit it to any other document management system, its structure intact. The traditional database community, on the other hand, focuses on ensuring modularity at the user's level, insisting on the interchangeability of systems. Given a set of data and an SQL query, any relational system and any hardware should provide the same result tables. The document community does not (yet) insist that queries posed on documents should produce the same result at each site, and the database community does not (yet) insist that there be a common interchange format for relational data.

Insistence on the independence of the data underscores the importance of markup to the document community, and illustrates why the development of standards for representing syntax has been considered an important task. This can be contrasted with the traditional database community, for whom it has been unimportant to settle on common formats for exchanging relational data, since much of this data resides in only one location, and is constantly being updated in any case. On the other hand, the traditional database community insists on the independence of their applications from underlying hardware and software, and developed abstract models that would ensure this independence, because the applications to be run were expensive to develop and can be more enduring than the data (for example, the application to compute payroll remains fixed, while individual salaries change).

Generalized markup has played a dominant role in awakening users to the virtues of data interchangeability. There are now several standards being developed to layer operational interchangeability on top of markup. It will be important to ensure that operational semantics are not driven by representational concerns.

Display. The third area of difference between text and traditional data is in display. Traditional database information is display-independent; it does not matter significantly what type of characters are used to print salary information, for example. One might dress up a report with pie charts and other graphs that illustrate trends more effectively than a simple table of values, but these displays are not themselves considered to be an essential part of the data. Display is much more important to document databases. For many documents, there is an original printed version whose display is information-bearing (for scholars at least). Thus, some aspects of the original layout of the text are important information. Even if the document never existed in paper form, however, display is important because the end result of searching a document database is the location of a text that is to be read, and reading is intimately involved with layout. Documents are in fact considered by some researchers to be interchangeable with displays; every display is a form of document, and every document is a form of display[4, 26].

Markup, whether presentational or generalized has long been the method for installing display information in online documents. Markup can be used to identify virtually any individual layout, and generalized markup is an effective way to describe many types of equivalence classes to parameterize the layout. On the other hand, markup's inheritance of the update properties of the content interferes with markup's ability to display features that change more rapidly than the update of the content allows. The alteration of positions caused by the insertion of markup complicates the coupling of the display to other processes. Markup's granularity limits its ability to describe details of an original document's appearance that are not properties of specific collections of characters (for example, a fold in the page or lacunae in manuscripts). And if documents are used as interfaces, the need to couple dynamic processes with the document will conflict with markup's static modelling, as described earlier.

A word should be said about the relationship between logical and presentational structure. Advocates for descriptive markup make a good case that it is difficult to extract logical structure from documents that are marked up presentationally, the main problem being that certain presentational effects are used to denote more than one logical element. However, it seems too much to conclude, as is usually done, that presentation must be derived from logical structure. First, there is no one logical structure to a document; the determination of logical structure must always be made with respect to some underlying theory, and there is no reason why this theory must be in accord with a document's presentation. Second, presentation has elements, constraints, and theories of its own, and it is reasonable for these to be managed in their own domain. Page layout, for example, is not derivable from the logical structures of documents.

The requirements of document databases exceed those of traditional databases. Current markup practices have led to the recognition of some of these requirements. It is important to note, however, that none of the requirements logically requires embedded structure. Markup is not a necessity for document management systems; rather, it is commonly employed for the conveniences we have come to expect and the experience that we have gained with its use. We should be aware of its tradeoffs so that we know when to use it, and when to use other structures.

7. Discussion.

Prior to the introduction of relational databases, traditional database systems were usually described operationally. Each database system was defined as a data structure and a set of operations on that structure. IMS databases, for example, provided a hierarchical structure and operations for navigating, querying and updating that hierarchy. With the advent of the relational model, the database community recognized the virtues of defining the organization of data in terms of mathematical abstractions, usually in terms of algebras. Mathematical abstractions have the advantage that they are separate from their representations, and are susceptible to theoretical investigation.

Markup-based document management systems are still largely defined in terms of the sequential data structure that is considered to be 'text'. Like IMS databases, they tend to see every text structure as a hierarchy, possibly defined by a grammar. As a result, these systems will suffer problems similar to those of databases not based on a mathematical model[6]. In particular, fundamental issues of identity and dependency will continue to be hard to separate from discussions about representations of data.

Markup and markup-based systems address a variety of issues simultaneously, including data interchange, data modelling, and storage representation. Markup's advocates see this combination as a strength. We suggest that the separation of these concerns is important in the long term for document management systems, just as it has been for databases. Markup is not a data model, it is a type of data representation. Since the formal properties of document management systems should be based on mathematical models, markup is unlikely to provide a satisfactory basis for document management systems.

Separating representational concerns from data modelling concerns may help to clear up current disputes about the true potential of markup standards for document management[29, 5]. Part of the problem is that advocates of markup claim advantages that are neither exclusive to markup nor particularly attendant on its use. Descriptiveness and portability, for instance, are not confined to certain types of markup, they are merely properties that intelligent use of markup share with many other conceivable approaches.

Sometimes the actual claims for markup-based systems are overstated; the claim that SGML results in portable documents, for example, falls afoul of the observation that it is possible to put angle brackets around *troff* tags, supply a simple document type descriptor, and thereby achieve an SGML-compliant document, without gaining any portability or descriptiveness for the information.† True portability requires not only that information be transportable from one machine to another, but that the semantics of that information be the same on either machine. SGML, in particular, claims to transfer no semantics, so it surely cannot guarantee portability.

Given an understanding of markup and its characteristics, what should we do next? One solution is to add a formal structure on top of markup-based systems. Such an approach is taken by Macleod *et al.*, for example, in their proposal to layer an applications program interface on SGML[19]. This reasonable approach has the advantage of leveraging existing markup-based systems. From the point of view of establishing solid foundations, however, this approach suffers in that it will be pushed to support existing standards, where the formalism might suggest a departure.

A second approach is to start with a mathematical abstraction for documents and try to develop a complete document management system top-down. In this approach, it is the abstraction that is most important, representations come later. This approach does not give results as quickly as the leveraged approach, but it holds more potential for clearly separating the formal issues from the representational ones. In the second approach, markup is only one of many potential forms of representation

8. Acknowledgements.

This work was financially supported by an IBM Canada Research Fellowship, by the Information Technology Research Centre of Ontario, and by the Natural Sciences and Engineering Research Council of Canada. We would like to thank Steve de Rose, Michael Sperberg-McQueen, Anne Brüggemann-Klein, and Robin Cover for their thoughtful comments on early drafts of this paper.

†This argument was provided to us by Steve de Rose.

References

1. David Barnard, Ron Hayter, Maria Karababa, George Logan, and John McFadden, "SGML-Based Markup for Literary Texts: Two Problems and Some Solutions," *Computers and the Humanities,* 22, pp. 265-276 (1988).

2. David Barnard, Lou Burnard, Jean-Pierre Gaspart, Lynne Price, C.M. Sperberg-McQueen, and Nino Varile, "Notes on SGML Solutions to Markup Problems," TEI MLW18, Text Encoding Initiative (April 16, 1992).

3. Lawrence A. Barsalou, "Ad Hoc Categories," *Memory & Cognition,* 11, 3, pp. 211-227 (1983).

4. Eric A. Bier and Aaron Goodisman, "Documents as User Interfaces," *Proceedings of the International Conference on Electronic Publishing, Document Manipulation and Typography,* pp. 249-262, Gaithersburg, Maryland (September 1990).

5. Pat Byrne, "SGML is a Data Exchange Standard, Not a Database," *Third Bellcore/BCC Conference on Electronic Document Delivery,* Parsippany, New Jersey (October 13-15, 1992).

6. E.F. Codd, *The Relational Model for Database Management: Version 2,* Addison-Wesley, Reading, Massachusetts (1990).

7. James H. Coombs, Allen H. Renear, and Steven J. DeRose, "Markup Systems and the Future of Scholarly Text Processing," *Communications of the ACM,* 30, 11, pp. 933-947 (1987).

8. E.W. Dijkstra, "My Hopes of Computing Science," *Proceedings of the 4th International Conference on Software Engineering,* Munich, Germany (September 17-19, 1979).

9. Albertine Gaur, *A History of Writing (revised edition),* Abbeville Press, New York, N.Y. (1992).

10. C.F. Goldfarb, "A Generalized Approach to Document Markup," *Proceedings of the ACM SIGPLAN/SIGOA Symposium on Text Manipulation,* 2, 1 & 2, pp. 68-73, Portland, Oregon (June 8-10, 1981).

11. Nelson Goodman, *Languages of Art,* Hacket Publishing Co. (1976).

12. Roy Harris, *The Origin of Writing,* Gerald Duckword & Co., London, England (1986).

13. Eric A. Havelock, *Preface to Plato,* Harvard University Press, Cambridge, Massachusetts (1963).

14. Text Encoding Initiative, *Guidelines for Electronic Text Encoding and Interchange,* Chapter 16, *Segmentation and Alignment*, Chicago, Illinois (January 21, 1993).

15. William Kent, *Data and Reality: Basic Assumptions in Data Processing Reconsidered,* North-Holland Publishing Co., New York (1978).

16. Brian W. Kernighan, "Issues and Tradeoffs in Document Preparation Systems," *EP '90, Proceedings of the International Conference on Electronic Publishing, Document Manipulation, & Typography,* pp. 1-16, Gaithersburg, Maryland (September 1990).

17. Chai Kim, “Retrieval Language of Social Sciences and Natural Sciences: A Statistical Investigation,” *Journal of the American Society for Information Science,* pp. 3-7, John Wilye & Sons, Inc. (January 1982).

18. David M. Levy, “Topics in Document Research,” *Proceedings of the ACM Conference on Document Processing Systems,* pp. 187-193, Santa Fe, New Mexico (December 5-9, 1988).

19. Ian A. Macleod, Brent Nordin, David T. Barnard, and Doug Hamilton, “A Framework for Developing SGML Applications,” *Proceedings of Electronic Publishing ’92,* pp. 53-63, Cambridge University Press, Lausanne, Switzerland (April 7-10, 1992).

20. Douglas L. Medin, “Concepts and Conceptual Structure,” *American Psychologist,* 44, 12, pp. 1469-1481 (December 1989).

21. Gordon Neal, “TEI Still Stunts Scholarship,” *Conference Abstracts and Programme for ALLC-ACH ’92,* pp. 186-191, Christ Church, Oxford, England (April 1992).

22. Brent Nordin, David T. Barnard, and Ian A. MacLeod, “A Review of the Standard Generalized Markup Language,” *Computers and Standards* (to appear).

23. Zenon W. Pylyshyn, “What’s in a Mind?,” *Synthese,* 70, pp. 97-122 (1987).

24. Vincent Quint, Marc Nanard, and Jacques André, “Towards Document Engineering,” *EP 90, Proceedings of the International Conference on Electronic Publishing, Document Manipulation, & Typography,* pp. 17-29, Gaithersburg, Maryland (September 1990).

25. Darrell R. Raymond, “Reading Source Code,” IBM Canada Laboratory Technical Report TR 74.070, IBM Canada, Toronto, Ontario (October 1991).

26. Darrell R. Raymond, “Flexible Text Display with *Lector*,” *IEEE Computer,* 25, 8 (August 1992).

27. Darrell R. Raymond and Frank Wm. Tompa, “Applying Database Dependency Theory to Software Engineering,” CS-92-56, Department of Computer Science, University of Waterloo, Waterloo, Ontario (December, 1992).

28. Frank Wm. Tompa and Darrell R. Raymond, “Database Design for a Dynamic Dictionary” in *Research in Humanities Computing I, Papers from the 1989 ACH-ALLC Conference,* ed. Ian Lancashire, Oxford University Press (September 1991).

29. Brian Travis, “SGML: It’s Not Just for Text Anymore,” *<TAG>: The SGML Newsletter* (July 1992).

30. Ben Zion Wacholder and Martin G. Abegg, *A Preliminary Reconstruction of the Unpublished Dead Sea Scrolls: The Hebrew and Aramaic Texts from Cave Four,* Biblical Archaeology Society, Washington, D.C. (1991).

31. Xinxin Wang and Derick Wood, “Tabular Abstraction for Tabular Editing and Formatting,” *Proceedings of 3rd International Conference for Young Computer Scientists,* p. Tsinghua University Press, Beijing, China (to appear 1993).

32. William Proctor Williams and Craig S. Abbott, *An Introduction to Bibliographical and Textual Studies, 2nd. Ed.,* Modern Language Association of America, New

York, N.Y. (1989).