MARKUP SYSTEMS AND THE FUTURE OF SCHOLARLY TEXT PROCESSING

Markup practices can affect the move toward systems that support scholars in the process of thinking and writing. Whereas procedural and presentational markup systems retard that movement, descriptive markup systems accelerate the pace by simplifying mechanical tasks and allowing the authors to focus their attention on the content.

JAMES H. COOMBS, ALLEN H. RENEAR, and STEVEN J. DeROSE

In the last few years, scholarly text processing has entered a reactionary stage. Previously, developers were working toward systems that would support scholars in their roles as researchers and authors. Building on the ideas of Bush [9], people such as Nelson [10, 27, 28] and van Dam [10] prototyped systems designed to parallel the associative thought processes of researching scholars. Similarly, Engelbart [16, 17] sought to augment human intellect by providing concept-manipulation aids. Reid developed Scribe, freeing authors from formatting concerns and providing them with integrated tools for bibliography and citation management [30]. Although only a small percentage of scholars was exposed to these ideas, the movement was toward developing new strategies for research and composition.

Since the introduction of inexpensive and powerful personal computers, we have seen a change in focus away from developing such new strategies toward finding ways to do the old things faster. This transition manifests itself in part through a change in models. Previously, developers worked with the model of scholar as researching and composing author. Recently, however, the dominant model construes the author as typist or, even worse, as typesetter.[1] Instead of enabling scholars to perform tasks that were not possible before, today's systems emulate typewriters. Granted, these electronic typewriters have built-in search and cut-paste facilities, but for the knowledgeable user, such systems offer only minor improvements and may be less powerful overall than some of the systems available 10 and 15 years ago.

There are a number of reasons for this trend. Probably most important, the transition from centralized computing to distributed computing began in business and industry, which remains the most compelling market for developers. Traditionally, such installations hire secretaries to type documents that are already substantially complete. The only tools required in such an environment are typewriter, scissors, and paste; or, now, their electronic equivalent.

Even in academia there is reduced impetus for more intelligent systems. Universities have their own business and administrative offices that make good use of business-oriented systems. Moreover, scholars often prefer these systems over the alternatives. Those who have access to more powerful systems rarely have the time to learn to exploit them fully, and many find them too complicated to use at all. This is quite understandable, since most text formatters on minicomputers and mainframes were developed under a model that is even more inappropriate than author-as-typist. Written by and for programmers, these systems often require quasi-programming skills, treating authors as programmer-typists. Most scholars experienced in computing are only too happy to escape such rough and poorly adapted systems for simple, handy, little programs that help them get text onto paper quickly. Lacking the concepts necessary to recognize major recent improvements and unaware of the possibilities for new strategies for research and composition, they hail this movement backward as a major advance in scholarly computing. Because this response comes even from the most experienced scholars, it carries weight with potential systems developers as well as with those who are just beginning to use computers. More and more

[1] Although there remain some important exceptions (e.g., [14, 31, 33]), this older, limited model of machine-scholar interaction has become dominant.

This work was supported in part by the Mellon Foundation. Although the issues addressed in this article bear on all electronic document development, we have chosen to focus on the domain that we know best: scholarly text processing.

scholars call for computing facilities that simply enhance their capacity to type; they exert pressure pulling the industry away from significant development. The consequence is an industry that is clinging to the past; scholars seek to do what they have always done, only a little faster.[2]

This shift in dominant models creates three major problems: First, the incentive for significant research and development in computing systems is disappearing, and a major portion of resources has been diverted into the enhancement of a minor portion of the document development process. Lacking the time to train themselves in other disciplines, many of the scholars who are setting the trends in text processing do not understand or value the possibilities. Moreover, the resources required for the development of sophisticated systems have been severely underestimated throughout the industry, and people have become impatient with the lack of immediately useful products. Thus, we see far more attention paid to keyboards, printers, fonts, displays, graphics, colors, and similar features than to the retrieval and structuring of information or even to the verification of spelling and grammar.[3] The development of tools providing new capacities has been replaced by safe and obvious enhancements of comfortable procedures. Second, developers and authors have lost sight of the fact that there are two products in the electronic development of a document: the printout and the "source" file. Currently, everything is directed toward producing the printout; the source is a mere byproduct, wasting valuable disk space, useful for little more than producing another printout of the same document. Proprietary formats and the lack of semantic and pragmatic coding make these files useless for sharing with colleagues or processing by intelligent systems. Finally, scholars' time and energy are diverted from researching and composing to formatting for final presentation. The authors of this article pay considerable attention to the quality of submissions and have typeset several books, but current systems tend to focus authors' attention on appearance all of the time, not just when the document is ready for submission.

Although there is no simple solution to all of these problems, major improvements in each can be made by converting to *descriptive* markup, which is already widely available. Briefly, the value of descriptive markup has gone unrecognized because authors and theorists believe the new document development systems—the supertypewriters—do not require any markup. Goldfarb, one of the primary developers of Generalized Markup Language (GML) [18], made clear, the advantages of descriptive markup over the usual *procedural* markup, but because no one has analyzed markup systems fully, users believe that no markup is even better than descriptive markup. As our analysis of markup systems makes clear, however, there is no such thing as "no markup." All writing involves markup. "No markup" really consists of a combination of *presentational* and *punctuational* markup. Moreover, of the competing types of markup, descriptive markup is actually easiest to learn, simplest to use, and best adapted to the process of composition. Finally, descriptive markup encodes the information necessary to fully develop the two products in the document development process: source files as well as printouts. Since the source files contain semantic and pragmatic coding instead of coding for formatting, they may easily be shared with colleagues, submitted directly to publishers, and processed by intelligent applications as they become available.[4]

In the first part of this article, we summarize the theory of markup systems and clarify the concepts necessary to properly evaluate the alternatives. We will now present the main arguments for the superiority of descriptive markup over other forms of markup.

MARKUP THEORY

Whenever an author writes anything, he or she "marks it up."[5] For example, spaces between words indicate word boundaries, commas indicate phrase boundaries, and periods indicate sentence boundaries. This fact is widely ignored; indeed, markup is usually treated as an unfortunate requirement of using electronic text-processing systems, that is, as something to be avoided. A careful analysis, however, reveals that authors regularly use two types of markup in their manuscripts: *punctuational*, for example, placing periods at ends of sentences; and *presentational*, for example, numbering pages. Thus, markup cannot be escaped because our writing system requires it.

These traditional, scribal types of markup clarify the written expressions. The markup is not part of the text or content of the expression, but tells us something about it. When we "translate" writing into speech (i.e., when we read aloud), we do not normally read the markup directly; instead, we interpret the markup and use various paralinguistic gestures to convey the appro-

[2] See Drucker [15] for a discussion of the costs of this tendency as well as of the failure to take calculated risks.

[3] At last count only four research groups in the country were attempting to develop sophisticated grammar-checking programs [1]. Spelling correctors and thesaurus programs have begun to appear in increasing quantities, but tend to be uneven in quality and are hardly the subject of intense scrutiny.

[4] This last point should not be underestimated: if we are to correct the current imbalance in allocation of resources, we must make it possible for newly developed products to process files created by all text-processing systems. Currently, not only do most files contain only paper-directed coding, they also occur in very incompatible formats. Even for a relatively trivial program, such as a spelling verifier, it can now cost a developer several months and thousands of dollars to develop the auxiliary programs required to process source files. Moreover, those who do not use the most common text processors are cut out of the market and do not have access to the new systems. The extra costs and the reduced market created by incompatible and nondescriptive coding constitute a considerable disincentive to the development of sophisticated systems.

[5] The following discussion is based on work by Coombs. This section owes major debts to Goldfarb [18] and to the recent ANSI-ISO Standard Generalized Markup Language (SGML) [2]. All descriptive markup examples are presented in SGML notation.

priate information. A question mark, for example, might become a raising of the voice or the eyebrows.

With the advent of text-processing systems came new types of markup and new types of processing. When prepared for reading, either on screen or on paper, documents are marked up scribally. But, when stored in electronic files, documents may be marked up scribally or with special *electronic* types of markup designed for processing by computer applications. One uses *procedural* markup to indicate the procedures that a particular application should follow (e.g., `.sk` to skip a line), *descriptive* markup to identify the entity type of the current token (e.g., `⟨p⟩` for paragraphs), *referential* markup to refer to entities external to the document (e.g., `&mdash;` for an em dash), and *metamarkup* to define or control the processing of other forms of markup (e.g., `⟨!ENTITY acm "Association for Computing Machinery"⟩` to define the referential markup `&acm;`).[6]

Types of Markup

Punctuational. Punctuational markup consists of the use of a closed set of marks to provide primarily syntactic information about written utterances. Punctuation has been under study for hundreds of years and is considered part of our writing system. Because punctuation is relatively stable, generally familiar to authors, and very frequent in documents, the usual expectation is that authors will punctuate their document files just as if they were typing.

Unfortunately, punctuational markup suffers from several deficiencies: The system is relatively complicated and subject to considerable stylistic variation. The authors of this article, for example, do not agree about the use of commas after sentence-initial adverbial phrases; in fact, composition instructors regularly disagree on such details. In addition to variations in usage, punctuation marks vary in appearance. For example, some people insist dashes should be separated from surrounding words by spaces; others claim there should be no such space.[7] Even where authors agree about appearance, there is variation across printing devices. Some devices provide the ability to distinguish open and close quotation marks; and some devices distinguish hyphens, en dashes, and em dashes. Finally, the punctuational markup system is highly ambiguous. For example, the period is used to indicate abbreviations as well as sentence boundaries. This ambiguity creates problems for text formatters, which often improperly treat abbreviations as sentence boundaries and add extra spaces. Authorial aids such as spelling and grammar correctors must perform considerable extra processing to disambiguate punctuation marks and often have to settle for the most likely alternative.

Recognizing the problems with punctuational markup, authors regularly use referential markup instead. For example, the source files for this article contain `&mdash;` instead of "---" or " --- " so the authors will be free to focus on the content and postpone stylistic negotiations until the end. Similarly, descriptive markup may be used to replace punctuation where the markup identifies a logical element. Short quotations, for example, are delimited by `⟨q⟩` and `⟨/q⟩` instead of quotation marks, enabling text formatters to output open and close marks or neutral marks, depending on the capacities of the display and printing devices. In addition, applications can quickly locate the quotations, for whatever purpose the author desires.

Thus, punctuation is not simply part of our writing system; it is a type of document markup that may vary and be replaced by other types of markup. Because the system is subject to stylistic variation, dependent on available printing devices, and highly ambiguous, we expect to see more and more punctuation replaced by referential and descriptive markup. On the other hand, we see little motivation for the complete replacement of punctuational markup. Not much is to be gained, for example, by replacing such standard punctuation as commas with referential markup. Publishers, or even text formatters, could use descriptive markup to determine whether a clause should be punctuated with a comma or a semicolon, but very few authors have sufficient training in syntax to mark up phrases and clauses descriptively. Consequently, punctuational markup will continue to be appropriate and need not be considered further in this article.

Presentational. In addition to marking up lower level elements with punctuation, authors mark up the higher level entities in a variety of ways to make the presentation clearer. Such markup—presentational markup—includes horizontal and vertical spacing, folios, page breaks, enumeration of lists and notes, and a host of ad hoc symbols and devices. For example, an author marks the beginning of a paragraph by leaving some vertical space and perhaps horizontal space as well. Occasionally, authors even number their paragraphs. Similarly, chapters often begin on new pages, may be enumerated in a variety of styles, and may even be explicitly labeled "Chapter."

Although authors have long performed presentational markup in their manuscripts and typescripts, most now prefer to have text formatters generate the most repetitive and error-prone markup. Pagination, for example, is usually automatic even in the most typewriter-oriented systems. "Local" presentational markup, however, such as centering lines, is still commonly per-

[6] Strictly speaking, some of our "electronic" markup occurs on manuscripts and typescripts in the form of proofreader's marks and typesetting directions, but such markup did not become prominent in the document development process until the advent of electronic systems. Goldfarb [18, p. 70] and SGML [2, pp. 90–91] consider our electronic markup "explicit" and our scribal markup "implicit." Actually, Goldfarb ignores presentational markup; SGML distinguishes "natural-language notation" (punctuational markup) from "formatted text notations" (presentational markup). Both types of scribal markup, however, have physical instantiation and, thus, are not really implicit. Moreover, both are part of a writing system and, thus, part of a medium for conveying natural-language expressions instead of part of the languages themselves.

[7] We have observed, for example, that Cambridge University Press favors the first style of dash; Oxford University Press, the second.

formed by the author, often with the assistance of editing commands. In WordStar, for example, one strikes the key sequence `Ctrl-OC` to center the current line.

Procedural. In many text-processing systems, presentational markup is replaced by procedural markup, which consists of commands indicating how text should be formatted. For example, one might mark up a long quotation as in Figure 1. The initial markup directs the text formatter to do the following (roughly):

(1) Skip three lines—the equivalent of double-spacing twice.
(2) Indent 10 columns from the left and 10 columns from the right.
(3) Change to single-spacing.
(4) Start a new page if fewer than two lines remain on the current page.

NO MARKUP
This example may look artificial, but ancient writing was often in such *scriptio continua*, with no interword spaces and little punctuation.

```
miltonexpressesthisideamostclearlylaterin
thetracticannotpraiseafugitiveandcloister
edvirtueunexercisedandunbreathedthatnever
salliesoutandseesheradversarybutslinksout
oftheracewherethatimmortalgarlandistoberu
nfornotwithoutdustandheatsimilarlywordswo
rth
```

PRESENTATIONAL

```
Milton expresses this idea most clearly
later in the tract:


    I cannot praise a fugitive and
    cloistered virtue, unexercised and
    unbreathed, that never sallies out
    and sees her adversary, but slinks
    out of the race where that immortal
    garland is to be run for, not with-
    out dust and heat.


Similarly, Wordsworth....
```

PROCEDURAL

```
Milton expresses this idea most clearly
later in the tract:
.sk 3 a;.in +10 -10;.ls 0;.cp 2
I cannot praise a fugitive and cloistered
virtue, unexercised and unbreathed, that
never sallies out and sees her adversary,
but slinks out of the race where that
immortal garland is to be run for, not
without dust and heat.
.sk 3 a;.in -10 +10;.cp 2;.ls 1
Similarly, Wordsworth....
```

DESCRIPTIVE

```
Milton expresses this idea most clearly
later in the tract:
<lq>
I cannot praise a fugitive and cloistered
virtue, unexercised and unbreathed, that
never sallies out and sees her adversary,
but slinks out of the race where that
immortal garland is to be run for, not
without dust and heat.
</lq>
Similarly, Wordsworth....
```

FIGURE 1. Forms of Markup

Obviously, this markup is specific to a particular text formatter and style sheet. Moreover, it is device dependent; the skip, for example, might well be changed to a value such as 18 points for a high-resolution printer.

Procedural markup is characteristically associated with batch text formatters, such as nroff/troff and TEX. Word processors like WordStar, however, supplement their presentational editing commands with "dot commands." For example, they use editing commands to center lines (`Ctrl-OC`), but include markup in the files for user-specified page breaks (`.pa`).

Descriptive. Under the descriptive system of markup, authors identify the element types of text tokens. In Figure 1 the tag `<lq>` identifies the following text as a long quotation, and the tag `</lq>` identifies the end of the quotation.

Authors who are accustomed to procedural markup often think of descriptive markup as if it were procedural and may even use tags procedurally. The primary difference is that procedural markup indicates what a particular text formatter should do; descriptive markup indicates what a text element is or, in different terms, declares that a portion of a text stream is a member of a particular class. When a text formatter generates a presentational copy of a descriptively marked-up document, it first reads in a set of rules, written in a procedural markup system, that establish what it should do for each occurrence of each element type. By adjusting this set of rules, the author (or support person) establishes a presentational markup design that will be executed automatically and consistently. Moreover, should there be reason to modify the design, only the rules will require editing; the document files remain intact. Not only will the author be relieved of painful and monotonous hours of mechanical editing, the text will not be exposed to corruption.

Most procedurally based systems provide macro facilities, which users have long exploited for descriptive markup (e.g., the `-ms` facility for troff), and even some of the primitives of such systems may be descriptive (e.g., the `.pp` "control word" for paragraphs in Waterloo SCRIPT). GML [18] provided a sound expression of the conceptual foundation for the systematic use of descriptive markup. Unlike ad hoc macro packages, GML is a descriptive language generally implemented on top of a clearly distinct, user-accessible procedural language. In addition, GML contributed “attributes” to descriptive markup languages, providing markup support for such essential functions as cross-references (which are automatically resolved by applications). Another influential system, Scribe, enforces the use of descriptive markup by eliminating procedural markup from the author's normal access to the system. Instead of tuning procedural markup to control the processing of descriptive markup, authors select “document format definitions” for various types of documents.

The Scribe approach has been widely emulated recently, but with moderate success at best. LATEX, for example, provides a high-level interface with TEX, which is designed to provide low-level typesetting control. Unfortunately, even the beginning LATEX user must think in terms of low-level markup. For example, contiguous quotation marks are to be separated by “`/,`”, which is a “typesetting command that causes TEX to insert a small amount of space” [21, pp. 13–14]. Similarly, a number of word processors (Microsoft Word, XyWrite, Nota Bene) have recently adopted Scribe's document format definitions under the metaphor of electronic “style sheets.” Nota Bene, for example, includes such editing commands as `use style block` for long quotations and has the ability to reformat all blocks when the style sheet is changed. Unfortunately, the style-sheet metaphor orients authors toward the presentation instead of toward the role of entities in the document. Thus, the block style might seem appropriate for any of a number of entity types, and nothing motivates the author to make distinctions that may be important later. Furthermore, style sheets tend to be an optional feature instead of a standard interface.

Referential. Referential markup refers to entities external to the document and is replaced by those entities during processing. We have already noted the use of referential markup for device-dependent punctuation (e.g., `&mdash;` for an em dash). Another characteristic use is for abbreviations, such as `&acm;` for “Association for Computing Machinery.” Referential markup might also refer to entities stored in a separate file or even on a different computing system.

Most text formatters that support procedural markup offer referential functionality through user-defined “variables” and file `imbed` or `include` commands. For the most part, however, referential markup is associated with descriptive markup systems, primarily SGML [2].

Metamarkup. Finally, metamarkup provides authors and support personnel with a facility for controlling the interpretation of markup and for extending the vocabulary of descriptive markup languages. Procedural and descriptive systems provide ways to define markup delimiter characters. In addition, procedural systems include such instructions as `define macro`, which are typically used to create descriptive markup representing a series of processing instructions. The procedural markup in Figure 1, for example, would typically be included in macros with names such as `quo` and `quoend`. Applications that process GML, such as Waterloo SCRIPT, also provide markup to define tags, to specify valid and default attributes, and to indicate what instructions should be executed when the tag is encountered. Finally, in SGML, metamarkup appears in the form of “markup declarations,” of which there are 13 kinds.

All nontrivial systems support metamarkup, but most do not provide a suitable interface for nonprogrammers. One notable exception is the menu-oriented group descending from Xerox Bravo and Star; InterLeaf, for example, allows authors to create new tags simply by typing hitherto unknown identifiers in a dialogue box. Others have attempted to eliminate the need for metamarkup by providing complete referential and descriptive vocabularies, but such efforts are contrary to the spirit of human creativity.

Markup Handling

Briefly, markup is selected, performed, stored, and processed. Familiarity with particular systems often complicates the task of distinguishing the various types of markup. Authors perform markup in a variety of ways. They may type the markup almost as if it were text. They may strike function keys or select items from menus. In fact, the methods of markup performance are limited only by the ingenuity of applications developers in using input and display devices. Although at any time there may be a tendency to associate a particular type of performance with a particular type of markup, the relationship is merely historical and provides no basis for characterizing or evaluating the types of markup.

Markup must be stored someplace, but there are no relevant limits on how it is stored. Moreover, nothing prevents a system from eliciting one type of markup and storing another. XyWrite, for example, elicits presentational markup, but stores procedural markup. When the author, who has access to editor commands but not to the markup language, specifies that text should be centered, XyWrite records the appropriate procedural markup around the text in the file and centers the line in the editing display. In a similar situation, WordStar simply centers the line; the surrounding blanks are not differentiated from the text either on screen or in the file. Thus, when evaluating their markup systems, authors must examine what is stored as well as what they see.

There are currently three major categories of markup processing:

(1) reading (by humans),
(2)' formatting, and
(3) open-ended (including formatting).

Presentational markup is designed for reading. Procedural markup is designed for formatting, but usually only by a single program. Descriptive markup is moderately well suited for reading, but primarily designed to support an open class of applications (e.g., information retrieval).

Exposed, Disguised, Concealed, and Displayed

In "traditional" text-processing systems, authors type in electronic markup, and documents are formatted by separate applications. Recently, formatters have been integrated with editors, and we need another set of distinctions in order to fully characterize markup in editing interfaces.

Markup is *exposed* when the system shows the markup as it occurs in the source file, that is, without performing any special formatting. Exposed markup is typical in systems that consist of separate editors and formatters. Many of the so-called WYSIWYG ("what you see is what you get") programs do not so much format for editing as expose the scribal markup they elicit and store. Such systems typically expose any electronic markup they elicit as well. WordStar, a sophisticated example of this category, exposes the "new page" command `.pa`, but also displays a line of hyphens to represent the page break.

More sophisticated systems often process electronic markup and then *disguise* it behind a special character. XyWrite and Nota Bene, for example, display a "delta" so that authors can locate and edit markup. Such systems usually have the capacity to expose the markup as well. Other systems (Xerox Bravo and Star, MacWrite) *conceal* electronic markup entirely. One system (Janus [11]) exposes descriptive markup on one monitor and conceals it on the other.

Finally, systems have recently begun to *display* electronic markup; that is, an especially formatted representation of the markup in the source file is displayed along with the text. Etude and Interleaf, for example, format text for editing, but display descriptive markup in a margin at the left of the editing window.

Because scribal markup is not well differentiated from text, current systems simply expose it. In fact, no other approach makes sense. It can be profitable to view electronic markup in any of the four modes, however. Datalogics' WriterStation, for example, supports all four modes and allows authors to control the formatting of displayed markup.

To summarize, there are six types of document markup, but only three are in full competition: presentational, procedural, and descriptive. Presentational markup clarifies the presentation of a document and makes it suitable for reading. Procedural markup instructs a text formatter to "do X," for example, skip three lines, in order to create presentational markup. Finally, descriptive markup tells text formatters that "this is an X," for example, a long quotation. Normally, text formatters treat presentational markup in source files as if it were text; that is, no special processing is performed. Procedural markup, however, is processed according to the rules specified in the documentation for the system; and descriptive markup is usually mapped onto procedural markup. In addition, descriptive markup is well suited for processing by an open set of applications.

Development systems should provide maximum flexibility and support all four modes of markup viewing. The author-as-typist systems that we have seen embraced recently tend to elicit presentational markup and store presentational and procedural markup; consequently, they bind documents to particular devices and applications. Some integrated editor/formatters, however, support descriptive markup, which, as we will argue, best supports the process of document development and publication.

MAINTAINABILITY

As we point out in our initial characterization, descriptive markup eliminates maintenance concerns. The development of a scholarly article may take several months; a book may take several years. In this time, an author who is not using descriptive markup may have to modify the markup of document files for any of several reasons:

(1) The author learns new techniques or finds that current techniques cause problems.
(2) The computing environment changes.
(3) The style specifications change.

In the development of *A Pre-Raphaelite Friendship* [13], for example, the editors initially used backslashes (\) to control highlighting. Their text editor, however, had the unfortunate habit of throwing away backslashes, and their early printouts suffered from random underscoring. They learned to avoid this problem by using the underscore character (_) to control underscoring, but also had to edit all of the text that had already been entered. When the book was typeset, they discovered the underscore character was also used as an en dash by their system (but only when typesetting). Consequently, when in the scope of an underscore command, en dashes were taken as underscore controls; instead of 1982–1986, the text formatter produced 1982*86*. Thus, the editors again had to edit all of their files and change the underscore characters to pound signs (#), and hope that no further conflicts would occur and that they would not introduce errors into the text during the process of making the changes. Had they used descriptive markup for highlighted phrases, these maintenance problems would not have occurred.

Similar problems occur whenever the author or the installation changes the computing environment. When FRESS (File Retrieval and Editing System) users at Brown University learned FRESS would no longer be supported, authors either spent hours converting their files to the new format (Waterloo SCRIPT) or accepted the possibility of "losing" their data.[8] Even updates in the current text formatter often require modifications in files. Changing to a new printer may require modifications. In fact, almost any change in the computing environment poses a threat to one's files if they contain procedural or presentational markup.

Finally, formatting specifications may change during the process of document development. For example, the Modern Language Association (MLA) recently pub-

[8] FRESS was directly based on the prototype system HES (Hypertext Editing System) [10].

lished a new style sheet. To give a sense of the consequences, we need consider only one change. The previous *MLA Handbook* [24, p. 23] specified that block quotations be "set off from the text by triple-spacing, indented ten spaces from the left margin, and typed with double-spacing (single-spacing for dissertations) but without quotation marks." Accordingly, many manuscripts include the procedural markup shown in Figure 1. The new edition of the *MLA Handbook* [25, p. 49] specifies that block quotations be "set off from your text by beginning a new line, indenting ten spaces from the left margin, and typing it double-spaced, without adding quotation marks." This modification immediately renders much of the markup obsolete, and now authors must locate all long quotations and delete `.sk 3 a`. Because the markup encodes formatting procedures instead of element categories, however, the occurrence of `.sk 3 a` cannot be taken as an unambiguous indication that an element is indeed a long quotation. Thus, authors cannot take advantage of global change facilities, but must inspect every occurrence of `.sk 3 a` and determine whether or not the current element is a long quotation. The conversion process will be tedious at best, and there is always the risk of corrupting the text. In addition, there is no guarantee MLA will not change its style sheet again, requiring further markup maintenance.

Such maintenance problems would not be reduced by using presentational markup. In fact, updating might be even more difficult. With procedural markup, one has specific character strings, such as `.sk 3 a`, that may be located with normal editing facilities. Presentational markup, however, may not be directly locatable. Some editors require a series of relatively advanced commands or the use of regular expression grammars to locate blank lines, for example. Moreover, simple editing facilities cannot distinguish, for example, a series of 5 blank spaces (for a paragraph indent) from a series of 5 blank spaces contained within a series of 10 blank spaces (for each line of a quotation). Thus, locating presentational markup accurately often requires the services of a powerful macro language as well as the ability to program.

With descriptive markup, properly tagged source files never require modification, and there is no such thing as markup maintenance. A long quotation, for example, remains a long quotation, despite changes in presentation style or even changes in processing systems. To modify the action taken for long quotations by a text formatter, one need edit only the program's "rule" base. This localization of maintenance can save numerous hours of editing, protects files from corruption, and makes it practical to have a single local expert perform necessary updating of a shared copy of the rule base.[9]

DOCUMENT PORTABILITY

The ability to "port" or send one's documents to other scholars and to publishers has always been a major concern of scholars. When typewriters ruled the industry, we ported our documents in the form of typescripts and photocopies. Since there were no alternatives, people were generally satisfied with this procedure.

In the last five years, however, more and more authors have shelved their typewriters and converted to electronic document development. Now we can send documents from our homes across the continent and around the world, often receiving acknowledgment of receipt within a few hours. Our colleagues, with our source files on their own machines, can use programs to search for keywords and can integrate our contributions into collaborative documents, free of the normal retyping or cutting and pasting. Moreover, publishers can use our files as a source for typesetting, eliminating the need for rekeying documents; and once the rekeying process is eliminated, so is the danger of textual corruption as well as the need to read proofs.[10]

Unfortunately, current text markup practices make this exchange a rarity. Although we have the technology for electronic transfer, we lack the markup standard necessary to guarantee that each recipient can process the documents prepared by any author. In fact, the compatibility problem is so severe that publishers often choose to rekey documents that have been submitted in electronic form, and sometimes do so without notifying the authors, who are left with a false sense of security about the integrity of their texts. As several publishers have pointed out to us, keying in documents is a simple, well-understood task requiring the services of people who are paid minimum wages. File manipulation, however, requires the services of personnel with programming skills, paid appropriately, and does not appear to offer sufficient gains to outweigh the risks of converting to a new process.

Descriptive markup provides an immediate solution to document incompatibility. Any document with accurate and rigorous descriptive markup can be ported from one system to another. This is true because descriptive markup guarantees a one-to-one mapping between logical elements and markup. Thus, element identifiers may be changed simply by performing global changes in an editor. For example, one might convert the markup for a prose quotation from `.quo` to `<lq>` and `.quoend` to `</lq>`. In the worst case, syntax differences may be resolved by trivial programs.

Recognizing this fact, representatives of publishers and of organizations with large publishing costs have joined in an effort to establish an industry-wide standard based on descriptive markup. In its *Electronic Manuscript Project* [4, p. 7], the Association of American Publishers (AAP) found that descriptive markup "is

[9] Rigorously used, electronic style sheets may provide the same solution to maintenance problems. Since style sheets are presentation oriented, however, we expect that authors will not make all of the necessary distinctions. The "block style," for example, might temporarily provide appropriate formatting for many different entities: prose quotations, poetry quotations, theorems, definitions, examples, etc. Later, it may be necessary to distinguish the elements, and the markup will need revision.

[10] Although most of the literature seems to assume the authorial reading of galley and page proofs will remain a part of the publishing process, we have eliminated that tiresome task from the production of books that we typeset ourselves. We see no reason for not retaining these advantages once publishers are able to typeset directly from authors' source files.

the most effective means of establishing a consistent method for preparing electronic manuscripts which can feed the publishing process." The AAP has endorsed the ANSI-ISO SGML and developed its first application. SGML, which is actually a metalanguage for generating descriptive markup languages, allows for considerable flexibility and customization. Authors who have been using descriptive markup will be able to turn their documents into SGML documents with little or no modification. Documents that have been prepared with presentational or procedural markup, however, will require extensive editing to conform with the new standard.

Advantages

Since people are generally reluctant to give up a technology they have learned, it is crucial that everyone be aware of what the industry has to gain by conversion to descriptive markup and, ultimately, to SGML. Consider this partial list of benefits:

(1) Authors will be able to share documents and collaborate with colleagues without the current concerns of incompatibility between text formatters and printing devices.
(2) Publishers will no longer have to rekey documents, eliminating an expensive and error-prone task.
(3) In many cases, the proofing process may be eliminated from the production cycle, saving considerable administrative costs for publishers and reducing the time required to get a document into print. Moreover, publishers will no longer have to negotiate with authors who want to make changes after the galleys have been set. For their part, authors will be relieved of the burden of proofreading documents that were correct at the time of submission.
(4) Subsequent editions, revisions, or collections may be generated from the source files for a document; rekeying will no longer be necessary.
(5) Bibliographic information may be generated directly from the source files. This will reduce errors and make citations available almost immediately to users of on-line bibliographic databases. The time from submission of a text to entry in the literature of a field will be cut dramatically.
(6) Documents may be included directly in on-line databases for electronic publishing and full-text retrieval, which is another way of introducing them into the literature almost instantaneously.

Publishers and authors have already begun to demand these improvements in the publishing process. With the expenses of scholarly publishing rising continually, cost' containment will become more and more important, and authors will find properly marked electronic manuscripts more marketable than other electronic manuscripts and typescripts.

Alternatives to Portability

Four alternatives to document portability have been proposed, but they provide partial solutions at best. The alternatives include

(1) authors typesetting their own work and providing camera-ready copy;
(2) authors submitting device-independent page description files, in PostScript, for example;
(3) authors submitting printouts, and publishers using Optical Character Readers (OCRs) to convert them to electronic form; and
(4) authors sending source files without descriptive markup, and publishers converting the markup with a special utility.

The first alternative involves authors excessively in the presentation process and distracts them from their role as authors. This procedure suffers from a number of severe problems: First, it should be clear that typesetting is a skilled task requiring special training in such concepts as typefaces, styles, and sizes; and leading, weighting, kerning, widows, rectos, versos, letterspacing, loose lines, and all the apparatus of professional designers. Moreover, most typesetting programs require either programming skills or extensive intervention. The nontechnical problems may be even more significant. Publisher's typesetting specifications usually suffer from a number of inadequacies that are customarily resolved through long-term relationships with local professionals. Thus, authors can expect to expend considerable time and energy clarifying specifications and resetting type. Moreover, like professional typesetters, author-typesetters may be held financially accountable for anything that is not set according to the publisher's specifications as well as any costs over the publisher's estimate. Finally, author-typesetters become subject to the tight deadlines of production cycles, which can interfere with their plans for teaching, scholarship, and administration.

The second alternative to document portability—providing page description files—still subjects authors to most of the problems of typesetting. In order to prepare page description files, authors must have the full typesetting specifications and ensure their files accord with those specifications.

The next alternative—submitting printouts to be read in by OCRs—relieves authors of typesetting problems, but does not significantly improve the production process. Although OCRs are becoming faster and more accurate, they are still expensive and error prone. Because of the need for operator intervention, there is little chance that proofreading could be eliminated from the production cycle. Moreover, OCRs have limited capacity to generate marked-up files from printouts. Current systems can generate some procedural markup, but none can distinguish a theorem from an axiom, for example, or even a section from a subsection. Thus, OCR-generated files still require the intervention of personnel trained to recognize and code textual elements. (Note also that even the operators will not be trained to make sophisticated distinctions, e.g., to distinguish an axiom from a theorem.) As character-recognition problems are solved, we might expect OCR manufacturers to concentrate on element recognition. Without explicit coding, however, automated element recognition will always be a haphazard task, and nothing could be more wasteful than to develop systems to recover knowledge that was thrown away when it could easily have been recorded in the source files.

Finally, there is a popular belief that publishers can convert authors' source files to their own formats by using special equipment. For example, according to *The Seybold Report on Publishing Systems* [20, pp. 37–38], "The Shaffstall communications/conversion system has gradually become known as a sophisticated tool capable of handling nearly anything that came along in the interfacing field." In their test of the Shaffstall 5000 XT, however, the reporters tried files from MacWrite 2, MacWrite 4.2, PageMaker, ReadySetGo, and Microsoft Word. They found the system processed the MacWrite 2 file "nicely," but failed with the others because "those programs handle the data differently." Obviously, the utility of such a system is jeopardized by every new version of every program. Moreover, such systems do not provide necessary element recognition; they simply generate source files with rudimentary procedural markup. Trained personnel are still required to identify each element type and mark it up appropriately. Finally, files are once again subjected to corruption.[11]

SGML has recently been criticized even by supporters of descriptive markup.[12] Above all, critics consider SGML too complicated both for authors and implementors. With its WriterStation, however, Datalogics has already demonstrated that sophisticated SGML tools can be developed and SGML document creation can be an intuitive process. Similarly, SoftQuad has produced an AAP text processor for the Macintosh. Sobemap has an SGML parser running under both Microsoft Windows and UNIX™ Version V.

This list of products is representative, not exhaustive, and development is actually just getting started. The Department of Defense is investing approximately $200 million per year in its SGML-based Computer-aided Acquisition and Logistics Support (CALS) initiative. Various other large government agencies, such as the Internal Revenue Service, are investing in SGML, and private organizations, such as McGraw-Hill, are planning to convert their systems fully in the near future. Some developers resent this pressure, especially from the government [19, p. 25]. The early successes mentioned above, however, will lead the way, and we can expect resistance to transform into serious development efforts.

Finally, SGML has been criticized for its lack of support for mathematics, graphics, and tabular material. SGML has metalinguistic properties, however, and AAP has already demonstrated that SGML applications can support mathematics and tabular material. We have no reason to believe the standard cannot support graphics through descriptive markup as well as through referential markup.

Portability Not Dependent on a Standard

We do not advocate waiting for SGML to become dominant. As we have illustrated, descriptive markup is vastly superior to both presentational and procedural markup. The superiority of descriptive markup is not dependent on its becoming a standard; instead, descriptive markup is the basis of the standard because of its inherent superiority over other forms of markup.[13]

Those of us who have converted to descriptive markup are already enjoying some of these outlined benefits. We have sent the source files for articles published in newsletters at Brown University to the University of Barcelona, which processed them without modification. Journals produced at Brown, such as *NOVEL: A Forum on Fiction* and *Philosophy and Phenomenological Research*, use descriptive tagging in the typesetting process and are preparing to accept submissions electronically. We can expect more and more journals and publishers to convert to this process when they realize the tremendous savings in administration and preparation costs. Ultimately, the industry-wide standard will accelerate this conversion to descriptive markup.[14]

Document portability promises significant reductions in costs and labor. The alternatives to document portability fail to address the need to share documents with colleagues and do not adequately address the problems of the production process. As AAP has recognized, descriptive markup provides the most complete and effective solution to the problem of establishing document portability. Many of the advantages may be enjoyed immediately, and those documents that have been descriptively coded can be updated easily if necessary as SGML becomes the industry-wide standard.

MINIMIZATION OF COGNITIVE DEMANDS

Basic Theory

All document markup takes place in three steps:[15]

[11] For more sophisticated translation efforts, see [22]. Note, however, that Mamrak et al. seek to support translation into and out of standards, not from one arbitrary format to another. Moreover, they chose a descriptive markup system for their prototyping: SGML.

[12] See [19, pp. 21–25] for a report of criticisms. Most of the information in this section on SGML activity has been derived from reports in [7].

UNIX is a registered trademark of AT&T Bell Laboratories.

[13] The opposite is true of another proposed standard: IBM's Document Content Architecture (DCA). Instead of declaring the logical status of textual entities, Revisable-Form-Text DCA declares formatting aspects and is presentation oriented. Thus DCA would not enable a publisher, for example, to switch from endnotes to footnotes or even to change to smaller fonts for block quotations without revising the source files.

[14] Our current incompatibility problems are not limited to document files. Every database management system, for example, has its own format, making it difficult to share information or even to switch from one system to another. Through his "Information Management System for Scholars" (IMSS) [12], Coombs has established that descriptive markup provides an appropriate and effective format for database portability. Not only can IMSS data files be ported to other database systems, scholars can now create bibliography and note files on a variety of machines and import them to IMSS to take advantage of its advanced functions. As they become aware of the possibilities, authors will consider such flexibility a requirement instead of a feature.

[15] This analysis generalizes on the three steps of procedural markup that Goldfarb discusses [18, p. 68].

(1) *Element recognition.* One recognizes the current element is a token of a particular type—paragraph, prose quotation, footnote, etc.
(2) *Markup selection.* One determines the markup that applies to the element type recognized in (1).
(3) *Markup performance.* One marks the element.

The best markup techniques require the least cognitive processing (ceteris paribus). We believe that steps (1) and (3) can be ignored. The first step, element recognition, is the same for all forms of markup: We have no pretheoretical motivation for positing that an author recognizes an entity is a footnote, for example, in one way when using descriptive markup and in another way when using other forms of markup. The third step, markup performance, has received the most attention, resulting in various forms of "keystroke minimization" and alternative interfaces (such as pull-down menus). Again, however, we have no pretheoretical motivation for believing that one form of markup is more susceptible to performance optimization than another.[16] A brief look at markup selection demonstrates that descriptive markup requires less cognition than either presentational or procedural markup. The differences have been represented visually in Figure 2.

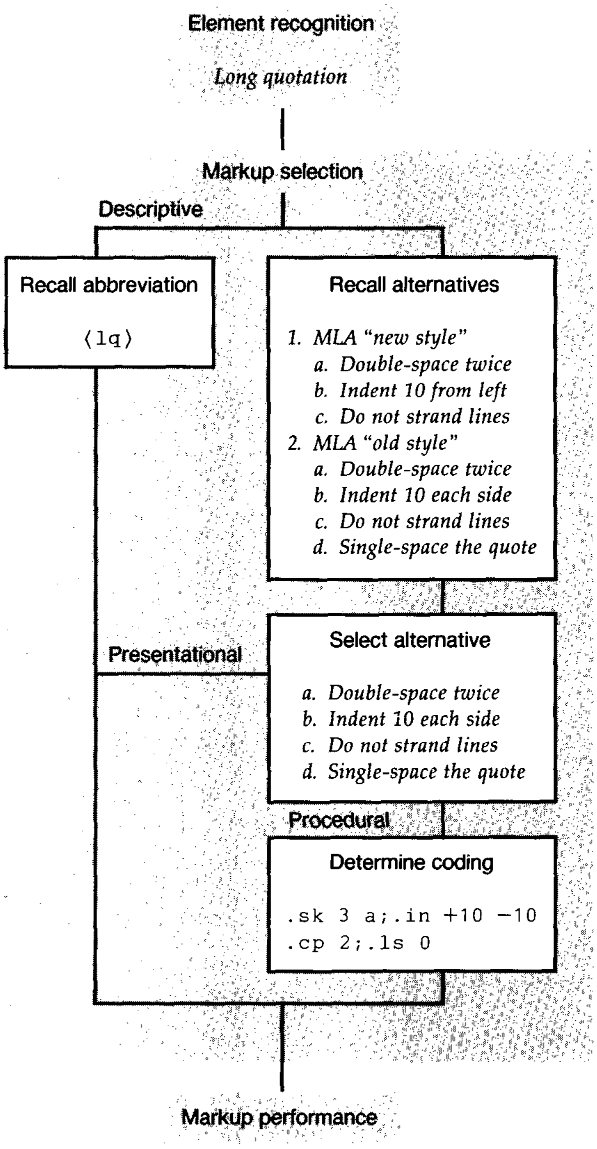

FIGURE 2. Markup Selection for a Long Quotation (roughly)

Presentational Markup

Under presentational markup schemes, markup selection requires an author to identify the set of typographical conventions for the current element type and then select the proper alternative.[17] This process is complicated by the fact that the relationship between typographical convention and text element is arbitrary and unstable. We have already discussed the maintenance problems that occur because of the instability of presentational markup conventions. When MLA changes its specifications for long quotations, for example, authors not only must update their files, they also must relearn the markup and are bound to go through a short period of confusion.

Even more important, the arbitrariness of presentational markup provides no support for recall and creates situations where there are an indefinite number of alternatives. High-frequency elements such as paragraphs may not present major recall problems; but infrequent elements such as citations require trips to style manuals. Similarly, elements with few alternatives such as paragraphs often do not require considerable reflection (although one may waver over whether a document is a personal letter or business letter and, consequently, whether paragraphs should be marked up by skipping lines or by indenting). The number of alternatives for higher level elements such as chapters, however, increases quickly with the flexibility of text formatters and printing devices. Publishers, who are at the extreme of flexibility, hire professional designers to select presentational markup. Authors, however, must decide for themselves how to mark up such elements and may spend numerous hours adjusting and readjusting fonts and skips.

One would naturally expect considerable simplification of the process after the initial selection of markup for a text element. Authors report, however, that their

[16] On the surface, it would seem that descriptive markup would provide for optimal markup performance. Whatever the interface, it should be possible to reduce the act of describing an element to a single edit action, but the process of identifying formatting procedures may well require multiple edit actions. Many editors, however, provide facilities for reducing a complicated series of editorial functions to a single keystroke. If the user thinks, for example, "By striking function key 1, I will double-space twice, indent 10 from the left, start a new page if necessary, and switch to single-spacing," then the user has performed relatively complicated presentational markup with as little effort as would be required to perform descriptive markup. The process of deciding how to mark up the element, however, will have been much more complicated, and we allow our argument to rest on that point.

[17] There is a whole range of possibilities for recall and selection that cannot be discussed here. This section should be considered suggestive instead of exhaustive.

memory fails and they often have to look back in a document to see how they have been formatting an element type. Moreover, they may well change their minds about the markup in the middle of the development of a document and interrupt the process of composition to reformat previously entered elements.

Procedural Markup

Markup selection is considerably more complicated for procedural-markup schemes. First, the author has to perform the same markup selection that is required for presentational markup. Then the author has to determine the procedural markup required by the target text formatter to create the selected presentational markup. Authors using procedural markup keep program documentation at their workplaces alongside style manuals.

Descriptive Markup

Descriptive markup reduces the process of markup selection to a single step falling naturally out of element recognition. Once the author has identified the element type, the proper markup may be determined simply by selecting the identifier that abbreviates the natural-language name for the current element type. In fact, the natural-language name may be used for the identifier and often is for infrequent elements such as "address." Such abbreviations as pq for "poetry quotation" simply provide for keystroke minimization.

Moreover, the markup selection for the termination of an element type (where necessary) is short-circuited by the convention of using a standard affix for the markup that was just selected to initiate the element. For example, once an author has selected ⟨pq⟩ as the appropriate markup to initiate a poetry quotation, markup for the end of the quotation is automatically ⟨/pq⟩, which may be arrived at through applying a standard and unchanging rule of affixation. Thus, descriptive markup is well suited for selection optimization.[18]

Descriptive markup minimizes cognitive demands by requiring little more than the normal linguistic processing already necessary to perform element recognition. Although the relationship between a text element and its natural-language name may be arbitrary, the relationship between the name and the descriptive markup is nonarbitrary. The tag for a paragraph might be p or para, for example, but it would not be sk or address. Moreover, the relationship between element name and descriptive markup is stable. MLA could modify its style sheet every day, but the descriptive markup for long quotations would remain ⟨lq⟩. Not only do authors not have to revise their markup, they do not have to learn new markup for every style variation. Thus, the composition process need not be interrupted by trips to style sheets, and authors do not have to look back into their files to see how they have been formatting a particular element type. In fact, they are freed from formatting concerns altogether.

Although authors generally believe that integrated editor-formatters simplify document development by eliminating the need for markup, our analysis reveals not only that such applications require markup, but also that the markup currently required by most popular systems is considerably more demanding than descriptive markup. Whether or not our analysis of the required processing is accurate in detail, it should be clear that (1) procedural markup selection requires previous presentational markup selection, and (2) because of the stable and nonarbitrary relationship between tag and element name, descriptive markup selection requires less cognition than other forms of markup. Finally, the additional cognitive processing required by presentational markup wastes energy that could be applied to the content of the document and may even cause a lengthy suspension of the composition process.

CONTENT ORIENTATION

One of the more subtle advantages of descriptive markup is it supports authors in focusing on the structure and content of documents. Both presentational and procedural markup tend to focus authors' attention on physical presentation.[19]

When one marks up text descriptively, one makes the logical structure of the text explicit. For example, the standard tag sets include "heading-level" tags, such as ⟨h1⟩, ⟨h2⟩, ⟨h3⟩, etc. Such markup clarifies both hierarchical and sequential relationships. In fact, some of us regularly begin our documents as skeletal outlines composed of these tags. As our documents expand, this markup of the structure helps us stay aware of the planned focus for each section. Moreover, as we will detail, we are able to use special applications to zoom in and out on the structure.

Unlike descriptive markup, presentational and procedural markup fail to support the writer in developing the structure; even worse, they distract from the content. As we have already discussed, the first step of marking up a document, element recognition, is the same for all forms of markup. The next step, markup selection, always involves some additional effort, but descriptive markup keeps this effort focused on the element and its role in the document. Presentational markup turns the author's attention toward typographic conventions and style sheets; and procedural markup leads even further away from the document toward the special markup required to make a particular text formatter produce the selected presentational markup.

There are a number of subtleties that we cannot address in detail here. The reader should bear in mind, however, that our argument addresses itself primarily to the process of markup selection. We have already noted it may be possible to perform complicated pre-

[18] For the sake of markup minimization, SGML introduces an "empty end tag" ⟨/⟩. Since this tag can be used only in certain contexts, this feature may complicate markup selection instead of optimizing it.

[19] Recognizing this fact, Reid [30] took presentational and procedural markup out of the hands of authors altogether. Only the Scribe Database Administrator has access to procedural markup and format definitions.

sentational or procedural markup as easily as it is to perform descriptive markup. In other words, any form of markup can be reduced to a single edit action. Conversely, a bad interface could make the performance of descriptive markup difficult and distracting.

In addition, we have raised, somewhat obliquely, issues of what authors view. We have suggested briefly that interfaces that expose or display descriptive markup will help authors focus on content and stay aware of structure. This is clearly an oversimplification. First, not all markup provides significant structural information or, better, information that is important to the author at the moment. Second, even descriptive markup can become so dense as to obscure the text. Furthermore, it might be in the author's best interests at any particular time to see, for example, the enumeration of items in a list instead of the descriptive markup. Ultimately, we have to conclude there is no simple relationship between viewing format and content orientation. An ideal system would provide authors with the ability to select among a number of different views of a file, and it should be possible to display the markup for some textual elements and to conceal the markup for others.

COMPOSITION ASSISTANCE AND SPECIAL PROCESSING

Using descriptive markup to identify the logical elements of a document not only simplifies composition, maintenance, collaboration, and publication, it also enables authors to apply a wide range of tools for composition assistance. This feature must be exploited if text processing is going to fulfill its original promise to significantly assist scholarly composition and become more than just improved typing.

Without the structural identifications provided by descriptive markup, the text is simply a "character string that has no form other than that which can be deduced from the analysis of the document's meaning" [18, p. 68]—and this is a deduction for which computers are entirely unsuited. The addition of content-descriptive markup changes this formless character string into a structured database of text elements, enabling the scholar to address those elements selectively and systematically. Two composition-assistance functions that exploit the ability to address text elements selectively through their markup are *alternative views* and *structure-oriented editing*.

Alternative Views of a Document

If a document is prepared with descriptive markup, then the text elements that are to be displayed in the editor can be globally specified by referring to the markup that identifies those elements. For instance, one could specify that all block quotations be concealed from view. Where once a lengthy quotation intervened between two sentences, there might now be only a flag indicating the number of lines concealed, or, if the user prefers, no flag at all: The sentences appear adjacent to one another. Similarly, all footnotes could be concealed, or all annotations, or all annotations of a certain class—say, all annotations about translation or by annotators other than the author. In this way the sections of the text that are of current interest are distinguished for attention, whereas those not of interest are hidden. Of course, at any point in the editing process, concealed text can be immediately and selectively disclosed for viewing.

Also useful are more specialized views, ones that are not just efficient for general composing and editing but finely adjusted to assist the scholar in *thinking* about the particular subject matter or topic at hand. Consider a specific example: One of the authors of this article writes papers on epistemic logic. His papers contain text-element tags such as *primitive* (to mark up text that explicitly introduces a term that is primitive within the axiomatic system being proposed), *definition* (for definitions of the system), *axiom* (for axioms of the system), and *theorem* (for theses that follow from the axioms). An editing utility that exploits this markup maintains a current list of every axiomatic element in the document. This list can be quickly consulted while editing—it is sometimes displayed in a concurrent editing window—and can be automatically updated and included as an appendix whenever the file is printed. Optionally, the listed axiomatic elements may be sorted and reviewed by *kind*—primitive terms, axioms, definitions, theorems—regardless of their order in the paper. As well as simply providing a quick reference to the current status of the axiomatic elements in the developing system, this utility is used to compare systems (sorted summaries of their axiomatic elements are put in adjacent windows) and to explore ideas about the relationships among elements. Similar techniques can be applied in other disciplines.

One source of the effectiveness of these viewing strategies is the simple juxtaposition of related sections of text for comparison. Another is the directed rearrangement of these regions—this is because frequently the chosen rhetorical structure of a document dictates a narrative order of text elements that is not the best sequence for the author's own thinking about the subject matter. In both cases the editing utilities described are using descriptive markup to overcome in the compositional environment the limitations imposed by the presentational requirements of the intended rhetorical form. This is exactly the sort of assistance a scholar should expect from the computer.

What is being accomplished here is something that cannot be performed effectively with conventional scholarly tools: Scattered sections of text that have been specified by certain content-related properties are being quickly presented for comparison and editing. With a conventional writing medium, like paper, only a laborious searching out of passages and cutting and rearranging could accomplish the same thing and even then with a significant likelihood of error. The time and energy required by this traditional physical method limit the number of rearrangements the author can afford to attempt and make the more experimental rearrangements especially impractical. With the computer-assisted approach, every hunch can be immediately explored, and viewing strategies can be progressively refined in accordance with the results of each trial, all with a minimal cost in time and energy. All these techniques require the structural information provided by descriptive markup; without that, computing scholars are little better off than they were with pencil, paper, and scissors.

Although the value of alternative views of a file as an intellectual tool has been described by a number of authors, surprisingly few scholars are taking advantage of these features.[20] Part of the problem is that the ability to present for display and editing alternative views of files is represented as a *feature* of particular experimental text-processing systems rather than a general *strategy* for text processing. However, by simply using a descriptive markup scheme and a general-purpose programmable editor any scholar can begin to take advantage of these tools immediately.[21]

Outlining and Structure-Oriented Editing

A special example of an alternative view is the *outline* (see Figure 3). Documents have a natural hierarchical structure: Chapters have sections, sections have subsections, and so on, until one reaches sentences, words, and letters. If the sections of a document are marked up as such, then it is frequently a simple matter to command a general-purpose editor to display only the section titles, presenting the scholar with a working outline. Furthermore, as section markup is keyed to section level, the author can easily have the outliner display the outline to any desired depth of detail, the lowest level of detail being the full text or, perhaps, the text with annotations and alternative versions. The level-by-level concealing and revealing of successive levels of detail may be controlled by a "zoom" function. One may also employ editing utilities to move hierarchical components of the document, as displayed in an outline view, and have the overall document structure adjust accordingly. For instance, one might directly move a section, complete with all its nested subsections and text, to another chapter. Or one might raise a section from a third-level section to a second-level section and have all of its nested subsection headings adjust their depth markup accordingly. Exploiting the natural hierarchical structure of text in this way has also been frequently recommended by text-processing theorists.[22] Although recently a number of powerful outlining applications have been marketed, again, these products are mostly specialized applications rather than general implementations of a strategy based on descriptive markup.

```
(h0) Background and New Concepts
     (h1) Introduction
     (h1) Markup Theory
          (h2) Types of Markup
               (h3) Punctuational
               (h3) Presentational
               (h3) Procedural
               (h3) Descriptive
               (h3) Referential
               (h3) Metamarkup
          (h2) Markup Handling
          (h2) Concealed, Exposed,
                 Disguised, and Displayed
          (h2) Summary
(h0) Advantages of Descriptive Markup
     (h1) Maintainability
     (h1) Document Portability
          (h2) Background
          (h2) Advantages
          (h2) Alternatives to Portability
          (h2) Portability Not Dependent on
                 a Standard
          (h2) Summary
     (h1) Minimization of Cognitive Demands
          (h2) Basic Theory
          (h2) Presentational Markup
          (h2) Procedural Markup
          (h2) Descriptive Markup
          (h2) Summary
     (h1) Content Orientation
     (h1) Composition Assistance and
            Special Processing
          (h2) Introduction
          (h2) Alternative Views of a
                 Document
          (h2) Outlining and Structure-
                 Oriented Editing
          (h2) Summary
     (h1) Conclusion
```

FIGURE 3. Outlining with Descriptive Markup

This kind of maneuver is an instance of a superior style of editing made possible by descriptive markup: *structure-oriented editing*. Descriptive markup allows the targets of editing operations to be not only characters, words, and lines, but also the actual text-element tokens, the regions of text that fall within the scope of a particular markup tag. For instance, one can execute operations such as "delete this footnote" or "move this quotation." Structure-oriented editing enables authors to address their documents at a level of abstraction appropriate to their authorial role; other forms of editing require the mediation of line displacements, line numbers, and marked regions. Thus, structure-oriented

[20] Important early discussions of alternative views of text are by Nelson [26, pp. 193–195], Engelbart and English [16, pp. 400–401], and Carmody et al. [10, pp. 288–300]. But, over a decade later, Meyrowitz and van Dam describe "the principle of multiple views" as "one that has been sorely underutilized in the hundreds of editors that have been created" [23, p. 405].

[21] Most of the functions described in this section have been implemented by the authors to meet some immediately perceived compositional need—they used only IBM's standard VM applications Xedit and Rexx.

[22] For an early discussion of outlining and structure-oriented editing, see [16]. For a selection of later work on structure-oriented editing and related topics, see the papers in [3].

editing minimizes cognitive demands and helps authors focus on content.

Composition-assistance features such as alternative views of a document and structure-oriented editing promise immense improvements in the effectiveness and productivity of the scholarly composition environment. The realization of most of these features depends on descriptive markup. Text processing with any other kind of markup will ensure that the anachronistic strategies of discarded technologies will continue to dominate computer-assisted document preparation.

CONCLUSION

Descriptive markup solves many of the problems that scholars face in document development. First, the process of marking up the text is simplified because the author's attention is focused on the content instead of on controlling a computer program or on the typography of the presentation copy. Second, maintenance problems are reduced to a few easily identifiable locations external to document files; updating for style changes is easy, and documents are not exposed to corruption. Third, as has been recognized by the AAP, descriptive markup provides the best means for establishing the industry-wide standard that we need for full document portability. Document portability enables authors to share files with colleagues and significantly reduces both the cost of publication and the nonauthorial demands on authors. Finally, in descriptively marking up a document, an author provides the semantic and pragmatic information necessary for such aids as alternative views on documents and structure-oriented editing. Without descriptive markup, only special systems with incompatible formats will offer even a portion of the authorial support that scholars have a right to expect from their computers.

In the end, it should be clear that descriptive markup is not just the best approach of the competing markup systems; it is the best imaginable approach. Currently, acceptance of descriptive markup is being retarded by authors' desires to retain familiar technologies and practices and by developers' use of proprietary formats to lock users into their products. Equipped with the basic concepts from the general theory of markup systems, however, authors quickly recognize the superiority of descriptive markup. As this awareness spreads, we can expect significant improvements in the quality of scholarly computing.

Acknowledgments. We are indebted to Mary Elizabeth McClure for stimulating our thinking about descriptive markup and to Richard A. Damon, Elli Mylonas, and David Durand for many helpful discussions. We would also like to thank Robert J. Scholes and Maurice Glicksman for their roles in acquiring the support necessary to research and develop this article.

Further Reading. For the seminal ideas on sophisticated authorial support systems, see [9], [10], [16], [17], [26], and [28]. Some important recent discussions are presented in [14], [31], and [33]. A number of good papers on document development and structure-oriented editing, including Goldfarb's important 1981 paper on GML, are contained in [3]. This volume also contains a useful "Annotated Bibliography of Background Material on Text Manipulation" compiled by Reid and Hanson. Reference [23] is the standard survey of editing systems and provides considerable information about formatters as well. Finally, the AAP and SGML standards are documented in [2] and [5].

REFERENCES

Note: References [6], [8], [29], and [32] are not cited in text.

1. Alexander, G.B. Computer aids for authors and editors: A natural extension of word processing and typesetting? *Seybold Rep. Publ. Syst. 13*, 10 (Feb. 13, 1984), 3–21.
2. American National Standards Institute. *Information Processing—Text and Office Systems—Standard Generalized Markup Language (SGML).* ISO 8879-1986 (E), ANSI, New York, 1986. (First edition: Oct. 15, 1986.)
3. Association for Computing Machinery. *Proceedings of the ACM SIGPLAN-SIGOA Symposium on Text Manipulation.* ACM, New York, 1981.
4. Association of American Publishers. *Electronic Manuscript Project: Task 1 Report.* Aspen Systems, Rockville, Md., 1984.
5. Association of American Publishers. *Standard for Electronic Manuscript Preparation and Markup.* Electronic Manuscript Series. Association of American Publishers, Washington, D.C., Feb. 1986.
6. Association of American Publishers. *Author's Guide to Electronic Manuscript Preparation and Markup.* Electronic Manuscript Series. Association of American Publishers, Washington, D.C., May 1986.
7. BDS. *⟨TAG⟩ The SGML Newsletter.* BDS, Sterling, Va.
8. Beach, R., and Stone, M. Graphical style—Towards high quality illustrations. In *SIGGRAPH 83 Conference Proceedings.* ACM, New York, 1983, pp. 127–135.
9. Bush, V. As we may think. *Atl. Mon. 176*, 1 (July 1945), 101–108.
10. Carmody, S., Gross, W., Nelson, T.H., Rice, D., and van Dam, A. A hypertext editing system for the /360. In *Pertinent Concepts in Computer Graphics*, M. Faiman and J. Nievergelt, Eds. University of Illinois Press, Urbana, Ill., 1969, pp. 291–330.
11. Chamberlin, D.D., et al. JANUS: An interactive system for document composition. In *Proceedings of the ACM SIGPLAN-SIGOA Symposium on Text Manipulation* (Portland, Oreg., June 9–10). ACM, New York, 1981, pp. 82–91.
12. Coombs, J.H. Information management system for scholars. Tech. Memo. TM 69-2, Computer Center, Brown Univ., Providence, R.I., Dec. 1986.
13. Coombs, J.H., Scott, A.M., Landow, G.P., and Sanders, A.A., Eds. *A Pre-Raphaelite Friendship: The Correspondence of William Holman Hunt and John Lucas Tupper.* UMI Research Press, Ann Arbor, Mich., 1986.
14. Corda, U., and Facchetti, G. Concept browser: A system for interactive creation of dynamic documentation. In *Text Processing and Document Manipulation*, J.C. van Vliet, Ed. Cambridge University Press, Cambridge, Mass., 1986, pp. 233–245.
15. Drucker, P.F. *Management: Tasks, Responsibilities, Practices.* Harper and Row, New York, 1973.
16. Engelbart, D.C., and English, W.K. A research center for augmenting human intellect. In *Proceedings of the AFIPS Fall Joint Computer Conference* (San Francisco, Calif., Dec. 9–11). AFIPS Press, Reston, Va., 1968, pp. 395–410.
17. Engelbart, D.C., Watson, R.W., and Norton, J.C. The augmented knowledge workshop. In *Proceedings of the National Computer Conference* (New York, June 4–8). AFIPS Press, Reston, Va., 1973, pp. 9–21.
18. Goldfarb, C.F. A generalized approach to document markup. In *Proceedings of the ACM SIGPLAN-SIGOA Symposium on Text Manipulation* (Portland, Oreg., June 9–10). ACM, New York, 1981, pp. 68–73. (Adapted as "Annex A. Introduction to generalized markup" in [2].)
19. Integration and pagination: Long documents, proposals, books. *Seybold Rep. Publ. Syst. 16*, 16 (Apr. 27, 1987), 21–27.

20. Interfaces, media converters and OCR devices. *Seybold Rep. Publ. Syst. 15*, 18 (June 2, 1986), 34–39.

21. Lamport, L. *LATEX User's Guide and Reference Manual.* Addison-Wesley, Reading, Mass., 1986.

22. Mamrak, S.A., Kaelbling, M.J., Nicholas, C.K., and Share, M. A software architecture for supporting the exchange of electronic manuscripts. *Commun. ACM 30*, 5 (May 1987), 408–414.

23. Meyrowitz, N., and van Dam, A. Interactive editing systems: Parts I and II. *ACM Comput. Surv. 14*, 3 (Sept. 1982), 321–415.

24. Modern Language Association. *MLA Handbook.* MLA, New York, 1977.

25. Modern Language Association. *MLA Handbook.* MLA, New York, 1984.

26. Nelson, T.H. Getting it out of our system. In *Information Retrieval: A Critical Review*, G. Schecter, Ed. Thompson, Washington, D.C., 1967, pp. 191–210.

27. Nelson, T.H. *Comput. Libr.* (1974).

28. Nelson, T.H. *Literary Machines.* Nelson, Nashville, Tenn., 1981.

29. Nievergelt, J., Coray, G., Nicoud, J.D., and Shaw, A.C., Eds. *Document Preparation Systems.* North-Holland, Amsterdam, 1982.

30. Reid, B.K. A high-level approach to computer document formatting. In *Proceedings of the 7th Annual ACM Symposium on Programming Languages* (Las Vegas, Nev., June). ACM, New York, 1980, pp. 24–30.

31. Trigg, R.H., and Weiser, M. TEXTNET: A network-based approach to text handling. *ACM Trans. Off. Inf. Syst. 4*, 1 (Jan. 1986), 1–23.

32. van Dam, A., and Rice, D.E. On-line text editing: A survey. *ACM Comput. Surv. 3*, 3 (Sept. 1971), 93–114.

33. Yankelovich, N., Meyrowitz, N., and van Dam, A. Reading and writing the electronic book. *Computer 18*, 10 (Oct. 1985), 15–30.

CR Categories and Subject Descriptors: H.4.1 [**Information Systems Applications**]: Office Automation—*word processing*; I.7.0 [**Text Processing**]: General; I.7.1 [**Text Processing**]: Text Editing; I.7.2 [**Text Processing**]: Document Preparation; K.1 [**Computing Milieux**]: The Computer Industry—*standards*; K.2 [**Computing Milieux**]: History of Computing—*software*; K.6.3 [***Management of Computing and Information Systems***]: Software Management

General Terms: Design, Human Factors, Languages, Standardization

Additional Key Words and Phrases: Descriptive markup, document interchange, generic coding, structure-oriented editing, text manipulation

Authors' Present Addresses: James H. Coombs, P.O. Box 2382, Providence, RI 02906; BITNET: JAZBO @ BROWNVM; Allen H. Renear, Computing and Information Services, Box 1885, Brown University, Providence, RI 02912; BITNET: ALLEN @ BROWNVM; Steven J. DeRose, Dept. of Linguistics, Box 1978, Providence, RI 02912; BITNET: NICK @ BROWNVM.