An Object-based Taxonomy for Abstract Structure in Document Models

R. FURUTA*
Department of Computer Science, University of Maryland, College Park, MD 20742

A document preparation system is characterised not only by the features included in its implementation but also by **what** ***is implemented. In this report, we focus on describing the document's representation, as visible to the author – the user of a document preparation system. The characterisation for paper-based documents subdivides the document's representation into a set of interacting structures – a primary structure that describes the composition of primitive objects into higher-level objects, secondary structures that describes the relationships among objects in the document, and auxiliary structures that represent sources or sinks of information outside of the document itself. Current document preparation systems are examined, from the viewpoint of their representation, and extension to hypertext is considered.***

Received June 1989

1. DOCUMENTS AND STRUCTURES

One's first view of a document preparation system is generally its implementation – for batch-oriented systems the syntax of the language provided for describing the document; for interactive systems the user interface that permits document manipulation. A deeper characterisation of the system is one that describes the document representation – *what* is represented, rather than *how* it is implemented.

The goal of this article is to provide a framework for describing the document representation. We will focus on paper-based documents – i.e. documents that are intended to be distributed, archived, and read on paper. Even though the paper form is considered to be the 'natural' form of such documents, it is useful to recognise that the paper form is but one representation in the lifecycle of the document. Furthermore, it is unlikely that a particular printed version of the document is the *final* form of the document – numerous interim versions of the document may be printed during the author's creation and revision of the document, different printed representations may be required from a particular version of a document for use in different contexts (e.g. for publication as a journal article and also as a chapter in a book), a new edition of a document may be created after the document is released, and portions of a document may be re-used in later documents.

An important class of documents not included in this focus are those documents that are intended to exist *only* in electronic form. Later in this article we will expand the discussion to consider the characteristics of hypertexts – one group of documents that fall into this class. In this later discussion (see Section 5) we will be particularly interested in identifying the similarities and differences in representation between paper-based and hypertext documents.

In general, the action of electronic creation and modification of a document is called *editing*.† We use the term *author* to represent the person who is carrying out these editing functions. Specification of the document requires specification of the *content* of the document and of the *structure* of the document – i.e. specification of the words and diagrams that make up the printed document and specification of how the pieces of content are to be fitted together into a whole. We will call this representation of the document the *abstract representation*. The language that the author uses to describe the abstract representation is called a *markup language*.

At some point in time, the author wishes to create a paper version of the document. This process of converting the abstract representation into a *physical representation* of the document is called *formatting*. The physical representation may be oriented to a specific output device, or it may be described in more generic terms. For example, the $\TeX$ document preparation system[35] produces output in `DVI` format,[17] specified in terms of a prototypical typesetter of extremely-high resolution. Similarly, typesetter-independent **troff**[33] produces output for an abstract typesetter of known resolution which is assumed to be able to handle a small collection of higher-level text-oriented and line drawing-oriented operations.

The physical representation of the document is then converted into a *page representation* – a representation in the format expected by a specific device.‡ Preparation of this representation, an operation called *viewing*, may require simulation of the abstract output device defined by the physical representation in the language of the actual output device.

To this point, we have presented an intuitive notion of three distinct document representations: the abstract representation, the physical representation, and the page representation. The relationship between representations has been presented as a transformation from one representation to another. We note in passing that the transformations are one-directional. In many cases, techniques for definition of the general transformations in the other direction (from page to physical and from physical to abstract) remains an open research issue.

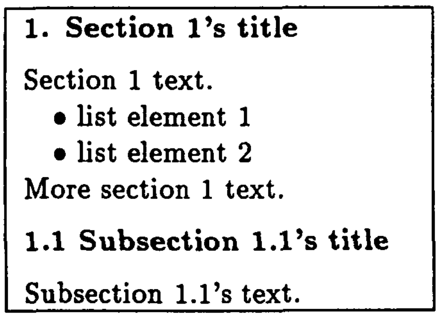
* Supported in part by a grant from the National Science Foundation, CCR-8810312.

† This model of the different representations that describe a document and the terminology used to define the operations on these representations are adopted from the model defined by Shaw.[16,49]

‡ Note that this representation may itself be in a page-description language such as PostScript.[1] Page description languages themselves represent an abstraction of the actual marking device – for example characteristics of the marking engine such as resolution may be abstracted out by the page description language.

We have not yet characterised the classes of objects related by the abstract representation. Indeed, this differs from representation to representation.

One important class of abstract representations is the *structured document*.[2] Such documents are based on the concept of 'generic markup', developed in IBM's GML,[21] popularised by Scribe,[46,54] and more recently, incorporated into the ISO SGML standard.[29] Intuitively, such document representations focus on identifying the local objects that make up the document, and not on the physical placement of elements onto the printed page. For example, a technical book might be described as a sequence of chapters, each chapter consisting of a sequence of sections, each section containing a sequence of subsections, each subsection containing a sequence of paragraphs, each paragraph a sequence of sentences, each sentence a sequence of words, and each word a sequence of characters.



1. Section 1's title

Section 1 text.

- list element 1
- list element 2

More section 1 text.

1.1 Subsection 1.1's title

Subsection 1.1's text.

Figure 1. A small sample document.

As example of such a markup scheme, consider the small document shown in Fig. 1. This figure shows a sketchy representation of a small document as it might appear when printed. One section of the document is shown, which first contains a title, then some text associated with the section and finally a subsection. The associated text contains an imbedded list, with list elements denoted by bullets. The subsection itself is titled, with some associated text.

Fig. 2 shows one generic markup representation of the small document. In this case, the syntax is that of LATEX.[37] LATEX commands beginning with the character '\'. Some of the commands in the example also have an argument, contained within the '{' and '}' brackets. This document is defined to be in the class `article`, thereby defining the commands that will be available for use in the remainder of the markup and also defining the effect that these commands will have on the displayed output. Note that specific details of the rendering of the commands are not specified in the markup; details such as the amount of space left before and after document elements, the typographic face and point size in which elements are to be displayed, the number associated with sectional divisions of the document, and the character that marks the elements of an itemised list in output ('.' in the example output).

The importance of the structured document is that emphasising the document's logical characteristics and de-emphasising the physical characteristics results in a representation with great flexibility. The specification can be mapped to multiple physical representations, thereby permitting easy and consistent modification to the appearance of the document and also easing re-use of the document's components. The identification of the document's logical components can ease transformation of the document, thereby assisting use of the document in other contexts. For example, abstracts of technical reports can be extracted for inclusion in a database, or sets of articles can be transformed into listings in the author's vitas.*

An equally common class of abstract representations is that based on markup describing the desired appearance of the objects comprising the document (this is sometimes called a description of the layout structure of the document). Such representations are particularly found in lower-level, implementation-oriented markup languages such as those defined by **troff**[33] and TEX.[35]†

```
\documentstyle{article}
\begin{document}

\section{Section 1's title}

Section 1 text.

\begin{itemize}
\item list element 1
\item list element 2
\end{itemize}

More section 1 text.

\subection{Subsection 1.1's title}

Subsection 1.1's text.

\end{document}
```

Figure 2. Generic markup representation of the sample document.

Fig. 3 shows a markup in **troff** for Fig. 1's small document. The **troff** commands used in this example are the document's content. Command lines are marked by the '.' on the margin, which is followed by the two-letter command name. Arguments to the command follow on the line. The command on the first line of the markup changes the line length to 3 inches. The next command switches the font to 'Bold'. After the section's title line is specified, the font is switched back to 'Roman'. The following `.sp 1` command leaves a line of blank space, and the `.in 2em` command changes the left margin of the output. Information such as the numbering associated with sectional division of the document and the character marking itemised list elements in specified explicitly in the markup (the `\(bu` in the markup specifies the '•' character on output).

The ODA representation[30] simultaneously applies two abstract representations to the document's content: a logical structure that defines the composition of document objects into successively larger logical components (as in the generic coding example above) and a layout

* A recent rediscovery of the advantages of the structured document representation can be found in Coombs, et al.[10] The structured document is described in more detail in a recent book,[2] which also includes discussion of current research areas.

† Similarly, Coombs *et al.*[10] identify three forms of markup: descriptive, corresponding to generic markup, procedural, corresponding to markup oriented to describing the appearance of the displayed objects, and presentational, corresponding to the appearance of the printed page.

structure that defines the composition of document objects into successively larger physical units: formatted blocks and then into pages.

Even more general application of sets of abstract representations has been proposed, see for example, Hamlet's[22] proposal which applies multiple logical structures to the document's content. Open research questions arise when multiple abstract representations are associated with a document's representation, for example whether mechanisms can be developed to permit simultaneous transformation of all the representations associated with the document. An interesting sub-question raised in recent research are the mechanisms for transforming documents defined by grammatically constrained structures so that they correspond to a different grammatically constrained structure.[15] A related question is the transformation among markup representations.[40–43]

```
.11 3in
.ft B
1. Section 1's title
.ft R
.sp 1
Section 1 text.
.sp 1
.in 2em
\(bu list element 1
.br
\(bu list element 2
.in
.sp 1
More section 1 text.
.sp 1
.ft B
1.1. Subsection 1.1's title
.ft R
.sp 1
Subsection 1.1's text.
```

Figure 3. Physical markup representation of the sample document.

2. MARKUP STRUCTURE

The taxonomy we will describe in the following section is based on a characterisation of the structure of the document's *markup representation*. In using a document processing system, an author specifies the content of the document and also imbeds *markup* specifications that direct the subsequent formatting transformation. Consequently, the document's markup representation is the representation of the document that the author sees and manipulates when specifying the document's structure and content. In essence, the markup representation is the description of the document's abstract representation manipulated by the author.

The characterisation of interest to us is one that focuses on the object and object relationships identified by the markup and not one that focuses on the particular *syntax* used to represent the markup. As a simple example of the syntactic issues, consider the differences between a markup scheme that flags commands with a special character located at a specific position on an input line, as in **troff**, and one that reserves a character to delimit the beginning of an imbedded command, as in Scribe and LᴬTᴇX. Such syntactic issues are of great practical importance to the author, but do not affect the object-based representation of the document.

As noted in the preceding section, commonly found abstract representation of the document include those that represent the logical relationships of the document's objects and those that represent the physical appearance and arrangement of those objects. Thus while the taxonomy is object-oriented, the specific objects identified may be either logical or physical in class.

3. TAXONOMY

The document representation is formed from a set of interacting constituent structures.* In this section, we will first examine the characteristics of such a constituent structure and then consider the broader categories into which the constituent structures that comprise the representation of a paper-based document may be placed.

Before we turn to a more detailed description of constituent structure characteristics and classification, it will be useful to develop an intuitive notion of the manner in which the component parts are combined into the document representation. The document description can be divided into two major parts: the definition of the document's content and the definition of the document's structure. The document's *content* presents the 'meaning' of the document – the words, figures, and the document elements that are read. These fragments of content are represented as a set of atomic objects (e.g. blocks of text, line drawings, tables, etc.). The document's *structure* specifies the *form* of the document, describing how the atomic objects are composed into higher-level structures (e.g., into sections or into pages of text), and describing the inter-relationships between the different atomic objects (e.g. a cross-reference, which specifies a relationship from the source of the reference to the designation of the reference).

Focussing on the document's structure, we note that two quite different kinds of relationships are defined on the content: the composition relationship that specifies the ordering of the atomic objects and how these atomic objects are combined into higher level objects, and the specification of interrelationships that establish a connection from one part of the document to another. Instead of describing the document's structure as a whole, therefore, we instead separate out the structures that describe each of these separate relationships. We call each of these structures a *constituent structure*, and define them separately. Each constituent structure is defined over some partitioning of the document's content into atomic objects. However, the partitioning appropriate for one constituent structure (e.g. composition into higher-level objects) may not be at the appropriate level of granularity for use in another constituent structure (e.g. composition into pages). Indeed, while an appropriate atomic object for use in one constituent structure might be a character string, the appropriate atomic object in a different constituent structure might be the individual characters of the string, or indeed sub-parts of those characters (i.e. the strokes that comprise the character). Describing the structure of the atomic

* The taxonomy presented in this article is a refinement of a more informally described representation that I have presented in earlier discussions.[14, 18, 20]

(1) Form of the constituent structure.
 (*a*) Minimum addressable unit(s) – the minimum nameable unit in the structure. Is the set of minimum addressable units static, or can new units be defined by the author?
 (*b*) Identifiable components and structures within the minimum addressable unit.
 (*c*) Higher level structure(s) that associated units together. Are units ordered, unordered, or partially ordered by the higher level structure? Are there constraints on higher level structures (e.g., grammatically specified constraints)?
 (*d*) Semantics associated with the constituent structure. Does the constituent structure represent composition (containment) of units, does it define an ordering of units but not composition (syntactic relationship), or does it symbolise a semantic relationship among the units? If it represents composition, what units are defined by the higher level structure(s)?

(2) Mappings between constituent structure semantics and document concepts.

(3) Representations of structure in markup of the document.
 (*a*) Representation of constituent elements in markup specification of the document.
 (*b*) Representation of constituent structure in markup specification of the document.

(4) Representations of structure in presentation of the document.
 (*a*) Visible presentation of constituent elements in printed document.
 (*b*) Visible presentation of constituent structure in printed document (e.g., source and destination of structure may be shown).

(5) Relationships between units of this structure and units of other constituent structures (identify any shared units).

Figure 4. The characteristics of a constituent structure.

object (the *minimum addressable unit* of the constituent structure) is therefore included in the description of a constituent structure, as is the relationship between the structure of the atomic objects of the different constituent structures.

The constituent structures that define the composition of document components into higher-level components are collectively referred to as the document's *primary structure*. Each of the constituent structures describing interrelationships among document objects is considered to define a *secondary structure*. To this point, we have only considered secondary structures whose source and target are both within the document. In the more complete case, either the source or the target may be to an object that is outside of the document – for example, the citation of a bibliographic reference within the document may define a secondary structure whose target is to an entry within a separately maintained bibliographic database. The constituent structures describing such external objects are called *auxiliary structures*. Further discussion of these classifications will be found in Section 3.2.

3.1 Characterising the constituent structures

The description of the document's representation is formed by identifying the primitive objects that comprise the document and by characterising the different constituent structures that relate the primitive object to each other. A number of interacting constituent structures may be identified, and the primitive objects relevant to each may represent different partitionings of the document. In such cases, the specification of the document's representation will need to describe the correspondences between the different sets of primitive objects.

Fig. 4 lists the information that characterises each constituent structure and its relationship to the remaining components of the document's representation. The characterisation is divided into four major parts, each of which will be described in turn.

The first part of the characterisation describes the form of the constituent structure itself, and as Fig. 4 indicates, is further subdivided to describe the components of the structure and its semantics. The structure is defined by the higher-level constituent structure (item 1*c* in the characterisation) and by the primitive document objects that are addressed by the constituent structure (item 1*a*). For example, one possible representation of a structured document is as a constituent structure of a tree in which the leaves correspond to blocks of the content of the document (paragraphs of text, tables, etc.). A minimum addressable unit may itself be structured – e.g., a paragraph of text is composed of a sequence of sentences, each composed of a sequence of words and punctuation, and each word composed of a sequence of characters. (Such further structuring is described in item 1*b* of the characterisation.) Editing operations that change the content of the document will alter the components and structures contained within a minimum addressable unit. However, the minimum addressable units are treated as atomic entities with respect to the overall constituent structure.

To be meaningful the specification of the constituent structure (items 1*c* and 1*d*) must include an indication of the constraints and semantics associated with the structure. The legal set of combinations of elements in a document may be constrained, for example by a grammatical specification, as in SGML.[29] A fundamental difference in the semantics associated with the constituent structures that define a document's representation is whether the structure represents composition, sequencing, or interrelationship of elements of the document. For example, a tree representation of a structured document might associate a document's section with the root of a subtree, and a subsection with each of the children of the root. In this case, the constituent structure represents a composition, and one in which there is an ordering: the section is formed from a sequence of subsections. As a second example, a constituent structure may represent a cross-reference – perhaps a reference in one part of a document to the location of an element in a different part of the document. In this case, the constituent structure does not represent a composition, but instead specifies a relationship from one portion of the document to another (i.e. a directed arc from a *source* to a *target*). As a similar example, associating a bibliographic citation with a particular entry in a database also defines a relationship but not composition. As a final example, consider the case as in a programming language definition in which a recursive grammar production is used to specify a *sequence* of elements. In this case the non-leaf portions of the corresponding parse tree are not semantically meaningful – the ordering of the leaf portions of the tree encapsulates the meaning of the structure.

The children of a tree node may be ordered, unordered, or partially ordered (item 1*c*). The previous paragraph has presented a number of examples in which the children are ordered. Unordered children may represent document elements that are contained in the same portion of the document, but that do not naturally follow one other; for example a figure and the text of the document discussing the figure. The children may also be partially ordered. For example, consider the partial ordering defined by the relationship between a footnote and its reference in a printed document – the text of the footnote cannot appear before the reference nor after a subsequent footnote, but there is no further ordering between the text of the footnote and the text of the remainder of that document.

The second part of the characterisation identifies the correspondence (or lack of correspondence) between the constituent structure and document constructs. In the preceding discussion, we have, for example, informally drawn a correspondence between sections of a document and nodes in a tree. Identification of such correspondences is an integral part of the characterisation of a constituent structure.

An additional issue of importance here is to determine the overall characteristics of the identified objects – i.e. if the description is oriented to the logical structure of the document or to a physical description of the printed document. Lower-level description languages such as **troff** present a physically-oriented representation of the document while higher-level description languages such as Scribe present a description more oriented to the document's logical structure.

The third part of the characterisation describes how the constituent structure is represented in the markup that is associated with the document. The focus on the markup specification is intended to illustrate the mechanisms that the author employs to represent the minimum addressable units and the constituent-structure-defined relationships. Once again, recall that the description of these mechanisms can be separated from the *syntax* that is used in their specification. In general, there is a direct correspondence between the constituent structure and the markup specification. However, it is useful to distinguish those portions of the markup specification that describe one constituent structure from those intended to describe another.

The fourth part of the characterisation describes how the constituent structure affects the displayed content of the printed document – in other words whether the relationship defined by the structure is reflected when the document is printed. Recall that a directed arc from the source of a reference to the target of the reference represents the interrelationship of two elements in the document. Either the source, the target, or both may be represented in the printed document. For example, when the arc represents a cross-reference, the page number on which the target appears may be inserted at the source. As another example, some systems permit the conditional inclusion or text in the printed document depending on the value of attributes associated with the document objects. (Ilson describes such a mechanism within the Interleaf system.[27]) This is an example of a situation in which the *context* in which an object is found affects the display of its *content*. Such mechanisms are useful for defining different versions of a document tailored for a particular environment (for example, a version for each of the different operating systems supporting a software package).

The fifth part of the characterisation describes the commonalities between the units defined by this constituent structure and by other constituent structures. Overlaps in the primitive structures used in each of several constituent structures would be noted here. Consider, for example, ODA's[30] separately defined logical and layout structures. The two structures share a set of content portions. In this taxonomy, the ODA logical and layout structures would be defined separately. The basic logical and layout objects (each consisting of one or more content portions) could be identified as the minimum addressable unit, and the commonality of content portion used in each would be noted in this part of the characterisation of each of the structures.

3.2 Structures in the document representation

A generalisation that may be applied to the constituent structures that comprise the representation of a paper-based document is that the structures can be classified as belonging to one of three non-overlapping categories:

(1) primary structure
(2) secondary structures
(3) auxiliary structures.

We will consider each of these structures in turn.

Intuitively, the primary structure is the predominating structure representing the composition of higher-level document objects from more primitive ones. Commonly encountered data structures corresponding to the primary structure are a linear list of document objects, a tree of document objects, and a rooted directed acyclic graph with document objects at the nodes. The linear list orders the associated objects, but does not specify any additional structuring. The tree of document objects defines a hierarchical composition of objects, and if the order of the children is significant defines an ordering of the objects as well. The directed acyclic graph, like the tree, defines a hierarchy and may define an ordering, but unlike the tree permits the representation of shared objects – objects that appear at multiple points within the hierarchy.

Secondary structures represent relationships among the document's objects that do not directly affect the ordering of objects or the composition higher-level objects. As each secondary structure represents a different class of semantic relationship within the document, it is expected that multiple secondary structures will be defined. Note, however, that this does not prohibit different secondary structures from sharing the same underlying representation. It is the association with a conceptual document relationship that distinguishes one secondary structure from another. Commonly-found secondary structures include those represented by cross references and by references to a separate bibliographic database. The relationship between a document and its table of contents and the relationship between index and document may also be viewed as represented by a secondary structure. Note that in these example cases, it is likely that the source of the structure corresponds to a point in the document while the destination corresponds to a larger document object (e.g. section, page, etc.). It is also likely that the source and possibly the target of the relationship are represented within the content of the document, i.e. that a string is generated that characterises the target (the section number or page location, for example) and that string is inserted into the printed document at the point corresponding the the source.

The third category of constituent structures are the auxiliary structures. As just noted, the target of a secondary structure may be external to the document's representation (as is the case with the separate bibliographic database). The constituent structure of this external target is categorised as an auxiliary structure. In the most general form, an auxiliary structure may represent an arbitrary externally defined computation, with the result of that computation providing the characterising string that is placed at the source of the associated secondary relationship in the printed document. In addition to this representation of the auxiliary structure in the document, filters may be associated with the auxiliary structure to convert the structure, or portions of the structure, into part of the document (for example, the selection of references entries in a bibliographic database to generate the reference list that is included in the document).

4. APPLICATION OF TAXONOMY IN DESCRIBING EXAMPLE DOMAINS

In this section, we will use the taxonomy to characterise a number of document representations. In Section 4.1, we will focus briefly on two abstract representations that concentrate on the physical structure of the document. In Section 4.2, we will consider three different representations that might appropriately be thought of as implementations of 'structured documents'.

4.1 Physically structured documents

An uncomplicated, physically oriented document abstract structure may be found in the 'What You See Is What You Get' (WYSIWYG) document preparation systems, such as MacWrite for the Apple Macintosh.[3] The primary structure in MacWrite is formed from individual characters (the character is the minimum addressable unit here). Characters are composed into words, words into lines, and lines into pages. Interspersed with the lines are *rulers*, which permit the modification of line-oriented formatting characteristics, such as margins, tab settings, line spacings, and alignments (i.e. left-aligned lines, centred lines, right-aligned lines, and filled and justified lines). Inserted material, for example graphics, is inserted into the structure as if it were a line. Character-oriented stylistic specifications, such as font and size, can be specified for substrings. Automatically placed headers and footers are associated with the page. No secondary structures are defined by the MacWrite document representation.

A somewhat more complex physical abstract structure is **troff**'s[33] (**troff** is the Unix system's batch-oriented document formatter). Again, there is an orientation to the structures that will appear in the printed document in the markup representation, with a primary structure based on the character, composed into words, words into lines, and lines into pages. Relevant commands specify the font and character parameters in effect until changed, control the length, formation, and spacing of output lines, and describe page characteristics. The mapping of a sequence of words to lines may be affected by word hyphenation. Similarly, the mapping of the lines to the page may be affected by location traps (causing execution of a sequence of commands when that location is reached on formation of the output page). Horizontal and vertical location at which output is to be placed can also be adjusted.

Interestingly, a higher-level structure that is defined is the *environment*. Environments are independent from one another, i.e., they are not nested within one another, and retain settings for some of the appearance-related parameters. Switching environments restores those parameters to the values associated with the selected environment.

Conditional operators permit the context in which the objects are contained to affect the content of the displayed document (i.e. this is a reflection of a case in which the structure is represented in the presentation of the document). A macro facility permits definition of new commands based on the facilities provided by the primitively defined commands. It is worth noting that, within the framework of the taxonomy, this permits the implementation of a document representation with different characteristics – indeed the implementation of **troff**'s `-ms` and `-me` macros more closely fits within the classification of logically-structured documents rather than physically structured documents. (Similar relationships can be found between LATEX.[37] implemented in TEX[35] and GML,[21,15] implemented in Script.[24]) It is also appropriate to note that preprocessors such as the table-oriented **tbl**,[38] the mathematically-oriented **eqn**,[31] the picture-oriented **pic**,[32,33] and the bibliography-oriented REFER,[39] use the **troff** language as their target in their implementation of additional document objects.

A number of secondary relationships are defined in **troff**. Values can be placed and retrieved from registers, in essence defining a relationship with its source at the point of insertion and its target at the point of retrieval. Values associated with predefined registers can also be obtained, a secondary relationship with its source outside of the document but with its target at a point within the document.

4.2 Logically structured documents

In this section, we will focus on three different document processing systems that might reasonably be said to provide implementations of the 'structured document'. The three systems are Interleaf's Technical Publishing Software,[28,44] referred to as 'Interleaf' in the discussion. Brian Reid's pioneering Scribe system,[46,54] and Xerox's Tioga.[51,52] We will also consider issues raised by SGML.[29] Of these four systems, Interleaf and Tioga are interactive document-preparation systems while Scribe and SGML are batch-oriented document formatters.*

In inspecting the document representations of these systems, one immediate observation is that in all cases the minimum addressable unit associated with the primary structure is similar in the textual case: strings of text that correspond roughly to the paragraphs of the document. What distinguishes the representations from one another are the higher-level structures that tie the components together.

Consider once again the small sample document presented earlier in Fig. 1. Fig. 5 shows an abstraction of the primary representation associated with the Interleaf system. This markup representation encodes the document as a linear list of objects. Each object is tagged with the name specifying the class to which the object belongs. In implementation, the classification of an object controls the manner in which it is displayed (this named specification of the manner in which the object is to be displayed is called a *look*).† The representation is called 'pseudo-hierarchical' because the adjacency of two similarly named objects implies that they are related – indeed it suggests that they should be viewed as composed into a higher-level object. Such higher-level composition, however, is not reflected in the representation.

Inclusion of a limited amount of such composition results in a primary representation that corresponds to Scribe's, as illustrated in Fig. 6. Notice that the list elements have been composed into a higher-level object called 'list', thereby forming an *environment*. (In Scribe, most environments also can be nested within other

* There have been interactive systems that incorporate the same model as Scribe, however. Most notably are MIT's Etude[23,26] and IBM's Janus.[7,8]

† Looks were originated in Xerox's Bravo system,[36] although this variety of look was unnamed.

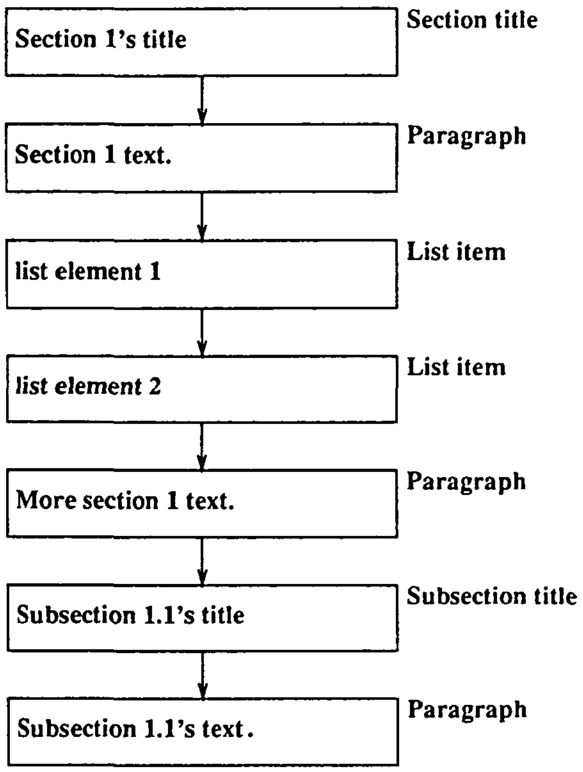


Figure 5. Linear, pseudo-hierarchical document representation.

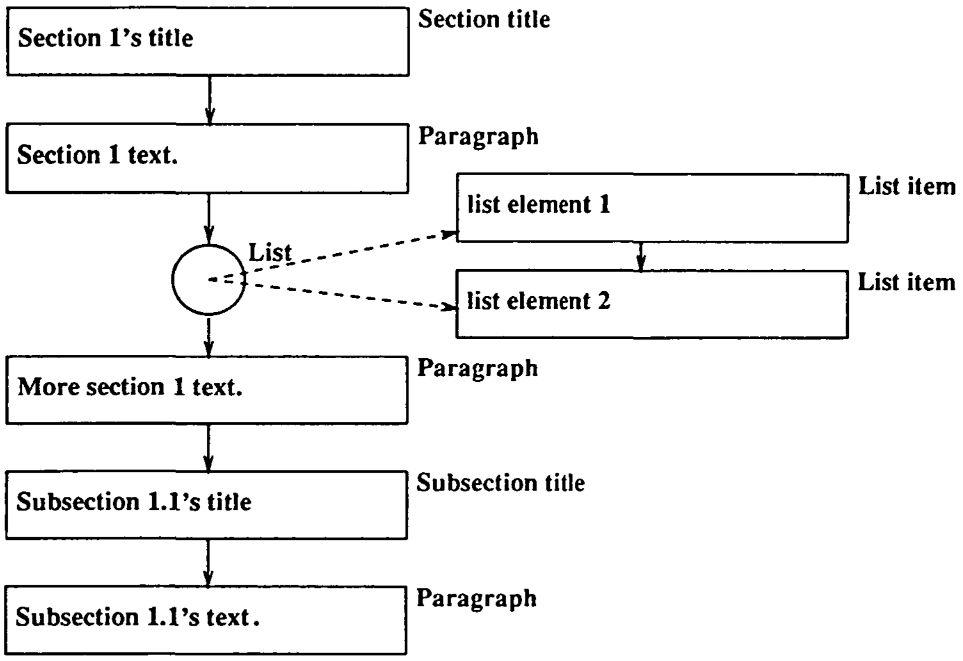


Figure 6. Limited environmental document representation.

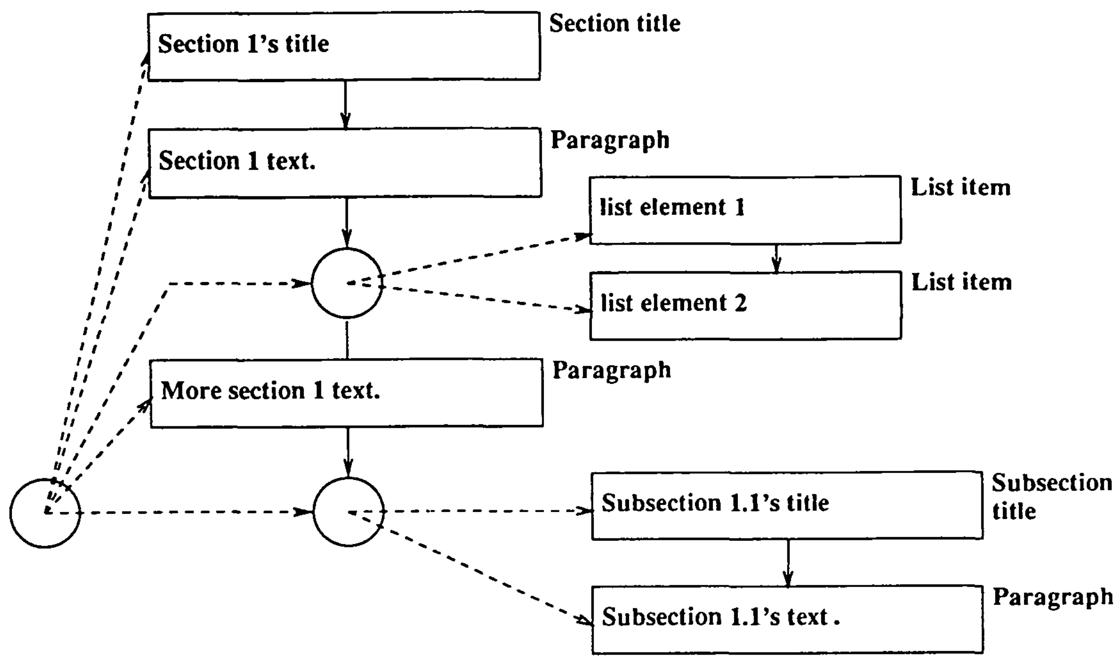


Figure 7. Hierarchical, unconstrained document representation.

environments.) However, the section is still represented as a linear sequence of title followed by content and the subsection is not nested within the section (indeed, there is no environment corresponding to the section).

Fig. 7 shows a representation of the sample document as a tree of objects, as found in Tioga. In the Tioga representation, the relationships between parts of the tree are unconstrained. The document's data is contained entirely at the leaves of the tree, and the internal nodes of the tree are not distinguished from each other. The Tioga implementation associates the name of a *look* with each of the characters in the text, specifying characteristics such as font and size, and associates the name of a *format* with each of the tree's leaf nodes, controlling elements such as horizontal and vertical spacing (format names are not shown in the figure). The looks and formats can be collected together into a *style*, which may be shared between documents. Modifying a definition within the style alters the appearance in all associated documents.

A distinguishing feature of a representation such as SGML's is that the object relationships are specified and constrained, for example by an associated grammar (the SGML DTD, in this example). When the constraints are satisfied, the representation of the sample document would resemble that of Fig. 7.

The document preparation systems discussed in this section also define a rich collection of secondary structures. Scribe's collection is representative. We have already mentioned cross references, in which the display of the source of the structure is based on some attribute of the target, for example a section number, the page number in the printed document on which the target appears, a figure number, the number associated with an element in an enumerated list, the number associated with an equation, the citation tag associated with a bibliographic reference, etc. In interactive systems such as Symbolics' Concordia,[56] the cross reference link may also be used as a traversal mechanism – the link may be followed from source to target, refocusing the display on the destination.

An interesting secondary structure is provided by Interleaf's *autonumber streams*. The autonumber stream represents a sequence of numbered entities. Autonumber tokens are associated with the stream, and are placed at points in the document. A token in the document is printed as the number associated with the stream (incrementing the number). Multiple autonumber streams can be active at a given point in time – interestingly this means that a stream's numbered objects (i.e. the objects containing autonumber tokens) do not have to be located contiguously, as is seen in environmentally-based schemes such as Scribe's and LATEX's. Interleaf *references* can be created to specific tokens, permitting use of the number and the page number it appears on in other portions of the document.

Inspecting the primary structure associated with different document representations helps us to better understand the reasons for the design and capabilities of the associated document preparation systems. As a broad generalization, the document representations that are associated with interactive document preparation systems are not as structured nor as constrained as those associated with batch-oriented systems. For example, Interleaf's primary representation shows very little additional structuring. Tioga's is structured, but the object relationships are not constrained, hence an indented representation of the document (with higher levels of indentation reflecting deeper levels of nesting) provides sufficient information about the higher-level structure of the document. Several interactive systems have been built using the Scribe model, but each incorporates special mechanisms to display the potentially nested environmental structure. One such system, MIT's Etude,[23,26] placed a separate window showing the nesting of the environments and their names to the side of the window displaying the formatted document. IBM's Janus[7,8] incorporated two completely separate windows, one showing the formatted document and the other showing the representation of the document in the GML markup language. Only the GML representation could be modified in the Janus implementation. Symbolics' Concordia, a more recent system than either Etude or Janus, imbedded a graphical representation of the environment's boundaries into the display, using indentation to further set off the body of the environment. Consequently, the representation shown to a Concordia author is a direct analogue to a Scribe-like markup file.

The reasons why a simpler representation is preferable in an interactive document preparation system are perhaps self-evident. The representation is more complex when there are numerous available objects, when the object interrelationships are strictly constrained or when the hierarchy relating objects is deep. Such effects must be presented to the author using the document preparation system, as the markup in the document prepared by the author reflects the document's representation.

However, there are also clear reasons for preferring a more complex document representation. A highly-constrained document representation permits specification of what elements are to appear in the document and what their ordering is to be. The constraints guarantee that any document created from the specification will not vary from the permissible format. This is a valuable property when the form of the document is specified by external sources – for example, documents that must meet requirements established by governmental agencies. A document template in an unconstrained system can establish these relationships but there is no way to prevent an individual author from modifying the template.

Highly constrained documents may not be as re-usable as less constrained documents. Consider the case when the object relationships are defined by a context-free grammer specification. Through use, a large body of document instances may accumulate based on a particular specification. If the specification changes, perhaps because of a change in the externally-established requirements or perhaps to correct an error in the specification, then one problem is how the instances corresponding to the older version of the specification are to be brought into compliance with the current version of the specification. Certainly a solution that simply detected the erroneous constructs would be insufficient, as the established base of document instances may be quite large. This problem is the focus of current research.[15]

5. HYPERTEXT

Hypertext systems define an interesting class of documents – documents that are intended to exist primarily in electronic form and are intended to be read through assistance from interactive computer software. The electronic form is the primary presentation of the hypertext document, not a statically defined paper form as in the documents discussed in the other parts of this article. Indeed the question of how a paper representation of a hypertext document can be generated is a current focus of research (the problem of *exporting* a hypertext document into paper form).

Hypertext is not a new idea. Vannevar Bush[5] in his 1945 article is generally credited for first expressing the principles on which current systems are based. As with document preparation systems in general, hypertext systems were an early application of computers. Two early systems from the late 1960's are HES[6,55] and NLS.[12,13]

Hypertext has gained much attention recently, particularly with the availability of PC-based systems. Recent surveys of the area include ones by Conklin[9] and Brown.[4]

In our analysis of Hypertext, we will focus on Hyperties,[45,50] which runs on the IBM PC. However, the analysis is not specific to Hyperties but is more generally applicable to currently-available hypertext systems.

A Hyperties hypertext is modeled by a directed graph. The content of the hypertext, called an *article* in Hyperties' terminology, is associated with the nodes of the directed graph. A *link* from one article to another is represented by a directed arc between the corresponding nodes in the directed graph. In Hyperties, the source of the link is represented by a highlighted string within the displayed content of the article. User selection of the string causes the link to be traversed. In the implemented user interface, the link traversal is a two step process: in the first step, the user sees a short summary description of the target. Confirming the selection causes a display of the destination article to replace that of the source.

A fixed set of buttons accompanies every display. 'Next page' and 'back page' buttons are used to turn the page of the currently displayed article. Also present is a button that permits return to the previously read article and one, named 'extra', that switches the display to permit use of an index, showing all articles defined in the database, a history list, permitting revisiting of a previously-seen article, and a string search mechanism, permitting selection of articles containing a reader-specified string.

Each article is further subdivided into three parts: a title, a brief description, and the body of the article. The body is described in a simple, physically-oriented markup language, and the link specifications are imbedded into this representation.

Applying the taxonomy to the Hyperties representation, we see that the article is an obvious choice for minimum addressable unit. The subdivisions of the article represent the identifiable components of the minimum addressable unit. Interestingly in comparison with paper-based documents, there is no structure defining object composition in Hyperties, and consequently the individual articles are isolated from one another when considering composition (i.e. the primary structure is a set of disconnected nodes).

The inter-article link structures fall within the definition of a secondary structure in which the source corresponds to the highlighted string at the point of reference and the target corresponds to a node. The index can be viewed as an additional secondary structure that contains references to each article in the database (it is not itself an article). The title of the article comprises the visible representation of the source of the link in the content of the index. Similar structures describe the history list and the string search list.

Our own investigation of hypertext structure has produced Trellis, a Petri-net-based model for hypertext.[47,48] This model augments the specification of the hypertext's structure and content with a specification of its *browsing semantics*. Browsing semantics are defined to be the manner in which the information in the hypertext is to be visited and presented. Such specification is often encoded in the *implementation* of the hypertext browsing (or reading) interface – i.e. in the behaviour of the software that implements the browser. Through use of the Petri net base, the Trellis model permits the author to specify the browsing semantics as part of the document in addition to the document's structure and content.

It is perhaps not surprising that specification of a document that is specialized for interactive manipulation should include details of how that manipulation may be carried out. Indeed, the notion of retaining reader-specified traversal paths as part of a hypertext is found throughout the history of hypertext: from Bush's original paper through more recent work such as Delisle and Schwartz's Contexts,[11] Zellweger's active paths,[57] and Trigg's guided tours.[53] While the document taxonomy describes the content and form of such interactively based documents, a complete description also requires specification of the semantics of that interaction.

6. CONCLUSION

Understanding the capabilities of a document preparation system requires more than identifying the features of the system's implementation details. It is also important to characterise the document representation used by that system. Selection of the document's representation involves tradeoffs – a physically structured representation may aid in the development of a system intended to produce one-shot, graphically complex brochures, but at the expense of making it hard to re-use parts of the document in subsequent brochures. A grammatically-constrained logical representation may enforce stylistic consistency and aid in specifying mechanisms to identify and transform document objects for other uses (such as databases). However, it may be hard to implement an interactive system that incorporates the grammatically based representation. (I discuss some of these points further in an earlier paper.[19])

Finally, we suggest that the structures and classification schemes developed for paper-based documents are also applicable to documents that exist only in electronic form, particularly hypertext. However, a complete characterisation of hypertext requires not only specification of what the document contains and how it is structured, but also specification of how it is to be read – the browsing semantics associated with the hypertext.

Acknowledgement

I would like to thank the National Institute of Standards and Technology's Electronic Publishing Laboratory for assistance in obtaining documentation for some of the systems discussed.

REFERENCES

1. Adobe Systems Inc., *PostScript Language: Reference Manual*. Addison Wesley, New York (1985).
2. J. André, R. Furuta and V. Quint, editors, *Structured Documents*. Cambridge University Press (1989).
3. *MacWrite*. Apple Computer, Inc. (1984).
4. P. J. Brown, Hypertext: The way forward. In *Document Manipulation and Typography*, edited J. C. van Vliet, pp. 183–191. Cambridge University Press (1988). (Proceedings of the International Conference on Electronic Publishing, Document Manipulation, and Typography, Nice (France), April 20–22, 1988.)
5. V. Bush, As we may think. *The Atlantic Monthly* **176** (1), 101–108 (1945).
6. S. Carmody, W. Gross, T. E. Nelson, D. Rice and A. van Dam, *A hypertext editing system for the /360*. Technical Report, Center for Computer and Information Sciences, Brown University, Providence, R.I. (1969). Also contained in *Pertinent Concepts in Computer Graphics*, edited M. Faiman and J. Nievergelt, pp. 291–330. University of Illinois, Urbana, Ill. (1969).
7. D. C. Chamberlin, J. C. King, D. R. Slutz, S. J. P. Todd, and B. W. Wade, JANUS: An interactive system for document composition. *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation, SIGPLAN Notices* **16** (6), 82–91 (1981). (The proceedings of the conference containing this paper are also available as *SIGOA Newsletter* **2** (1&2) (1981). This report was also issued as IBM Computer Science Research Report Number RJ3006 (37371), IBM Research Laboratory, San Jose, Calif., December 1980.)
8. D. C. Chamberlin, J. C. King, D. R. Slutz, S. J. P. Todd and B. W. Wade, *JANUS: An Interactive Document Formatter Based on Declarative Tags*. IBM Computer Science Research Report RJ3366 (40402) (1982).
9. J. Conklin, Hypertext: an introduction and survey. *Computer* **20** (9), 17–41 (1987).
10. J. H. Commbs, A. H. Renear and S. J. DeRose, Markup systems and the future of scholarly text processing. *Communications of the ACM* **30** (11), 933–947 (1987).
11. N. M. Delisle and M. D. Schwartz, Contexts – a partitioning concept for hypertext. *ACM Transactions on Office Information Systems* **5** (2), 168–186 (1987).
12. D. C. Engelbart and W. K. English, A research center for augmenting human intellect. *Proceedings, AFIPS Fall Joint Computer Conference* **33**, 395–410 (1968).
13. D. C. Engelbart, R. W. Watson and C. Norton, *The augmented knowledge workshop*. ARC Journal Accession Number 14724, Dtanford Research Center, Menlo Park, Calif. (1973). (Paper presented at the National Computer Conference, June 1973).
14. R. Furuta, V. Quint and J. André. Interactively editing structured documents. *Electronic Publishing: Origination, Dissemination, and Design* **1** (1), 19–44 (1988).
15. R. Furuta and P. D. Stotts, Specifying structured document transformations. In *Document Manipulation and Typography*, edited J. C. van Vliet, pp. 109–120. Cambridge University Press (1988). (Proceedings of the International Conference on Electronic Publishing, Document Manipulation, and Typography, Nice (France), April 20–22, 1988.)
16. R. Furuta, J. Scofield and A. Shaw, Document formatting systems: Survey, concepts, and issues. *ACM Computing Surveys* **14** (3), 417–472 (1982).
17. D. Fuchs. The format of $\mathrm{T_EX}$'s DVI files. *TUGboat* **3** (2), 14–19 (1982).
18. R. Furuta, Structured document models and representations. In *Issues in Generalized Text Processing: Lecture notes of a Short Course held in association with PROTEXT IV, the Fourth International Conference on Text Processing Systems*, edited J. J. H. Miller, pp. 1–14. Boole Press (1987).
19. R. Furuta, Complexity in structure documents: User interface issues. In *PROTEXT IV: Proceedings of the*

Fourth International Conference on Text Processing Systems, edited J. J. H. Miller, pp. 7–22. Boole Press (1987).

20. R. Furuta, Concepts and models for structured documents. In *Structured Documents*, edited J. André, R. Furuta and V. Quint, pp. 7–38. Cambridge University Press (1989).
21. C. F. Goldfarb, A generalized approach to document markup. *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, *SIGPLAN Notices* **16** (6), 68–73 (1981). (The proceedings of the conference containing this paper are also available as *SIGOA Newsletter* **2** (1 & 2) (1981).)
22. R. Hamlet, A disciplined text environment. In *Text Processing and Document Manipulation*, edited J. C. van Vliet, pp. 78–89. Cambridge University Press (1986). (Proceedings of the international conference, University of Nottingham, 14–16 April 1986.)
23. M. Hammer, R. Ilson, T. Anderson, E. J. Gilbert, M. Good, B. Niamir, L. Rosenstein and S. Schoichet, *Etude: An integrated document processing system*. Office Automation Group Memo OAM-028, M.I.T. Laboratory for Computer Science, Cambridge, Mass., February (1981). (Presented at the 1981 Office Automation Conference, 23–25 March 1981.)
24. *Document Composition Facility: User's Guide*. IBM Corporation (1980). Order number SH20-9161-1.
25. *Document Composition Facility Generalized Markup Language: Starter set reference*. IBM Corporation (1980). Order number SH20-9187-0.
26. R. Ilson. *An integrated approach to formatted document production*. Technical Report MIT/LCS/TR-253, M.I.T. Laboratory for Computer Science, Cambridge, Mass. (1980). (M.S. thesis.)
27. R. Ilson. Interactive effectivity control: Design and applications. In *Proceedings of ACM Conference on Document Processing Systems*, Santa Fe, New Mexico, December 5–9, 1988), pp. 85–91. ACM, New York. December (1988).
28. *Technical Publishing Software: Reference Manual*. Release 3.0, Apollo version, vols 1 and 2. Interleaf (1987).
29. *Text and Office Systems – Standard Generalized Markup Language*. ISO (1986). Document Number: ISO 8879–1986(E).
30. *Office Document Architecture*. International Standard Organisation (1986). Draft International Standard 8813.
31. W. Kernighan and L. L. Cherry, A system for typesetting mathematics. *Communications of the ACM* **18** (3), 151–157 (1975). Also available as Computing Science Technical Report No. 17, Bell Laboratories, Murray Hill, N.J. (revised 1977).
32. B. W. Kernighan. PIC – A language for typesetting graphics. *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, *SIGPLAN Notices* **16** (6), 92–98 (1981). (The proceedings of the conference containing this paper are also available as *SIGOA Newsletter* **2** (1 & 2) (1981).)
33. B. W. Kernighan, *A typesetter-independent TROFF*. Computing Science Technical Report 97, Bell Laboratories, Murray Hill, N.J. (1981).
34. B. W. Kernighan, PIC – A language for typesetting graphics. *Software – Practice and Experience* **12** (1), 1–21 (1982).
35. D. E. Knuth, *The TEXbook*. Addison-Wesley (1984).
36. B. W. Lampson, Bravo manual. In *Alto User's Handbook*, edited B. W. Lampson and E. A. Taft. Computer Science Laboratory, Xerox Palo Alto Research Center, (1978).
37. L. Lamport, LATEX: *A Document Preparation system*. Addison-Wesley (1985).
38. M. E. Lesk, *Tbl – A Program to Format Tables*. Computing Science Technical Report 49, Bell Laboratories, Murray Hill, N.J. (1976).
39. M. E. Lesk, *Some Applications of Inverted Indexes on the UNIX System*. Computing Science Technical Report 69, Bell Laboratories, Murray Hill, N.J. (1978).
40. S. A. Mamrak, J. Barnes, J. Bushek, and C. K. Nicholas, *Translation Between Content Oriented Text Formatters: Scribe, LATEX, and Troff*. Technical Research Report OSU-CISRC-8/88-TR23, Computer and Information Science Research Center, the Ohio State University (1988).
41. S. A. Mamrak and C. L. Joseph, *Translation for WYSIWYG Word Processors in Chameleon*. Technical Research Report OSU-CISRC-11/87-TR43, Computer and Information Science Research Center, the Ohio State University (1987).
42. S. A. Mamrak, M. J. Kaelbling, C. K. Nicholas and M. Share, A software architecture for supporting the exchange of electronic manuscripts. *Communications of the ACM*, **30** (5), 408–414 (1987).
43. S. A. Mamrak, M. J. Kaelbling, C. K. Nicholas, and M. Share, *Chameleon: A System for Solving the Data Translation Problem*. Technical Research Report OSU-CISRC-8/88-TR24, Computer and Information Science Research Center, the Ohio State University (1988).
44. R. A. Morris, Is what you see enough to get? A description of the Interleaf publishing system. In *PROTEXT II: Proceedings of the Second International Conference on Text Processing Systems*, edited J. J. H. Muller, pp. 56–81. Boole Press (1985).
45. G. Marchionini and B. Shneiderman, Finding facts vs. browsing knowledge in hypertext systems. *Computer* **21** (1), 70–80 (1988).
46. B. K. Reid. *Scribe: A Document Specification Language and its Compiler*. Ph.D. dissertation, Carnegie-Mellon University Computer Science Department, Pittsburg, PA (1980). Also issued as Technical Report CMU-CS-81-100.
47. P. D. Stotts and R. Furuta, Adding browsing semantics to the hypertext model. In *Proceedings of ACM Conference on Document Processing Systems* (Santa Fe, New Mexico, December 5–9, 1988), pp. 43–50. ACM, New York (1988). (An earlier version of this paper is available as University of Maryland Department of Computer Science and Institute for Advanced Computer Studies Technical Report CS-TR-2046 and UMIACS-TR-88-43.)
48. P. D. Scotts and R. Furuta. Petri net based hypertext: document structure with browsing semantics. *ACM Transactions on Information Systems* (1989). To appear.
49. A. C. Shaw. *A Model for Document Preparation Systems*. Technical Report 80-04-02, University of Washington, Department of Computer Science, Seattle, WA (1980).
50. B. Shneiderman. User interface design for the Hyperties electronic encyclopedia. In *Proceedings of Hypertext '87*, pp. 199–204 (1987).
51. W. Teitelman. A tour through Cedar. *IEEE Software* **1** (2), 44–73 (1984).
52. W. Teitelman. A tour through Cedar. *IEEE Transactions on Software Engineering* **SE-11** (3), 285–302 (1985).
53. R. H. Trigg. Guided tours and tabletops: Tools for communicating in a hypertext environment. In *Proceedings of Conference on Computer-Supported Cooperative Work* (Portland, Oregon, September 26–29, 1988), pp. 216–226 (1988).
54. Unilogic, Ltd. *Scribe Introductory User's Manual*. Unilogic, Ltd, Pittsburgh, fourth edition, April 1984.
55. A. van Dam, Hypertext '87 keynote address. *Communications of the ACM*, **31** (7): 887–895, July 1988.
56. J. H. Walker, Supporting document development with Concordia. *Computer*, **21** (1), 48–59 (1988).
57. P. T. Zellweger. Active paths through multimedia documents. In *Document Manipulation and Typography*, edited J. C. van Vliet, pp. 19–34. Cambridge University Press (1988). (Proceedings of the International Conference on Electronic Publishing, Document Manipulation, and Typography, Nice (France), April 20–22, 1988.)