ELECTRONIC PUBLISHING, VOL. 1(1), 19–44 (APRIL 1988)

Interactively editing structured documents

RICHARD FURUTA
Department of Computer Science
University of Maryland
College Park, Maryland 20742
USA

VINCENT QUINT
INRIA and the University of Grenoble
Laboratoire de Génie Informatique, B.P. 68
F-38402 St Martin d'Hères
France

JACQUES ANDRÉ
INRIA/Irisa-Rennes
Campus de Beaulieu
F-35042 Rennes Cedex
France

SUMMARY

Document preparation systems that are oriented to an author's preparation of printed material must permit the flexible specification, modification, and reuse of the contents of the document. Interactive document preparation systems commonly have incorporated simple representations—an unconstrained linear list of document objects in the 'What You See Is What You Get' (WYSIWYG**) systems. Recent research projects have been directed at the interactive manipulation of richer tree-oriented representations in which object relationships are constrained through grammatical specification. The advantage of such representations is the increased flexibility that they provide in the reusability of the document and its components and the more powerful user commands that they permit. We report on the experience gained from the design of two such systems. Although the two systems were designed independently of each other, a common set of issues, representations, and techniques has been identified. An important component of these projects has been to examine the** WYSIWYG **user interface, retaining the naturalness of their user interface but eliminating their dependencies on the physical-page representation. Aspects of the design of such systems remain open for further research. We describe these open research problems and indicate some of the further gains that may be achievable through investigation of these document representations.**

KEY WORDS Document preparation systems Structured documents Grammatically-defined generic document structures User interfaces Design experience

INTRODUCTION

An important stage in the development of a document preparation system is the design of flexible, interactive tools that aid an author in preparation of documents for publication. Although the printed page is a static, physical form, the document itself is a fluid, changing object. From the point of view of the author, document development follows a lifecycle, similar to the one that has been observed in the development of computer programs. The document is first created, then modified in preparation for publication, published, and then reused, either through re-publication in new editions or through reuse of portions in different contexts. The initial creation of the document is a cycle of design and refinement (writing and rewriting)—the document is specified and iteratively modified as syntactic and semantic inconsistencies are detected. Transferring the document to a publisher initiates a new cycle of modification to the document as the document is

0894–3982/88/010019–26$13.00

Received 1 November 1987
Revised 25 November 1987

proofread and edited to conform to the publisher's style. Publication of the document does not necessarily suspend its development. Re-publication of a technical document in a new edition will generally involve modification to the document's content to reflect advances in the subject area. Reprinting of a document by a different publisher frequently requires modifications to meet the differing stylistic requirements of the new publisher. Even if the document is never re-published, portions of it may be reused in new documents.

One goal in the design of a document preparation system is to identify a document representation that permits the flexible reuse of the document and of its components. One such representation is that of the document as formed from a collection of hierarchically-related components. The relationships between components are based on the document's logical structure and not the physical appearance of the components on the page. For example, the document may be described as consisting of a sequence of chapters, each chapter consisting of a sequence of sections, each section a sequence of paragraphs followed by subsections, and so on, until the document is described completely in terms of its basic component objects. This separation of the document's contents from its appearance enhances the ability to reuse the document and its parts—the same specification can be transformed into the different physical representations needed to meet specific formatting requirements. Consequently, such representations are insulated from changes to the formatting requirements. Such object-based document representations were originally associated with batch-oriented formatters and form the basis of document specification through generic markup.[1] Documents represented in such a fashion will be called *structured documents* in this paper.

A further goal is to represent the document entirely within the framework of the document preparation system. Documents contain not only text, but also tabular material, mathematical notations, line drawings, pieces of programs, bitmaps, and other objects. The document representation must permit specification of the wide variety of component objects.

Although computers have been used in the preparation of documents for publication for more than twenty years, developing the techniques that permit incorporation of structured document representations into interactive systems remains a topic of active research. In this paper, we report on common experience gained from two independent projects directed to development of such systems. In one, a prototype system (which we will call the **pedtnt**) has been defined to manipulate a document representation called the **tnt**[8,9]. This prototype has been used as a testbed for research into representation and manipulation of structured documents, but is not intended to be a production system. The second, called Grif[10,11,12], has produced a system that is approaching production quality. Grif has been developed in France at INRIA and the University of Grenoble.

The similarities and differences between the two designs serve to clarify the issues in the design of interactive document processing systems that manipulate structured documents. In the subsequent sections of this paper, we will first discuss some of these issues in general terms considering why interactive manipulation of structured documents re-

[1] IBM's GML[1,2]. originated the concept of generic markup and Reid's Scribe[3] popularized it. These and other early systems have been surveyed elsewhere[4]. Similar structures have also been incorporated into the SGML[5] and ODA ISO standards[6]. These and other recent developments are discussed in a forthcoming book that collects papers on aspects of structured document representation and use[7].

quires creation of new techniques, then we shall describe our solutions, and finally we present some of the problems that are still open and require further research.

GENERAL ISSUES

We associate three distinct representations of a document with a document processing system: the document model representation, the output model representation, and the display representation.[2] To this point in the discussion, we have concentrated on a document model representation which specifies the *logical structure* of the document. The prototypical representation of the logical structure is the book-oriented one sketched out already. However, it is relevant to note that other kinds of documents may also be described in the same terms, for example letters and forms. The particular objects that are identified will differ from one class of document to another, as will the objects' interrelationships, but the form of the structure that describes the the object interrelationships will remain the same and therefore the same mechanisms can be used to specify the different document classes.

The document model representation is transformed into the output model representation by a mapping called formatting. The output model representation describes the document's physical appearance. The physical appearance may be described, for example, in terms of a two-dimensional page space, representing the position of elements (such as individual characters) within that page space. This representation specifies the attributes that one commonly associates with printed documents—fonts, character sizes, line breaks, etc. The output representation is mapped to a display representation—in other words to a form for display on a particular medium (e.g., CRT or paper). This is called the viewing mapping.

Use of a document model that is based upon structured documents raises general questions. These questions focus first on the structured document representation itself. Additionally, questions arise on the means that associate the description of the document's physical appearance with the elements of the logical structure—in other words the means by which the transformation from document model to output model is specified. We will present these general issues in the remainder of this section, describing the solutions our systems incorporate later in the paper.[3]

STRUCTURED DOCUMENTS

The simplest document model represents a document as a sequence of printable characters interspersed with control sequences that modify the position or graphic aspect of the following characters. By contrast, the structured document is a high-level representation that is based on the logical organization of a document, but not directly on its graphical form. The simple model is the basis of many popular and widely used document preparation systems—for example the WYSIWYG editors. However, the structured document representation permits more generalized manipulations of the document.

---

2 These classifications are based on those defined by Shaw[4].

3 We note in passing that the solutions we propose here are not limited only to application in our systems. Such techniques are likely to be of use when applied to any representation incorporating a similar document model representation—i.e., one that is based on grammatically-specified, publication-oriented, structured documents. A notable member of this class of representations is that defined by the SGML standard.

In addition to enhancing the reusability of the document and its components, structured document representations allow the development of tools that perform auxiliary manipulations on the document and features that assist the author of the document. It is not only possible to print the document as if it were represented in the simple model, it is also possible to build tools that automatically extract components (abstract, keywords, or title, for example) for storage in an information retrieval system, or tools that translate the document into different formalisms, perhaps in support of document interchange.

Another advantage of this representation is that the editing system may present the user with powerful commands. According to the structure, the system may number (and renumber when necessary) all elements that need to be numbered, it may compute and update cross-references, and it may establish a table of contents or an index. All these features are already available with high-level batch formatters such as Scribe or LaTeX, as they use a similar level of abstraction for representing documents. The novelty is that these features are interactive and thus allow even greater functionality. For example, the user may use the cross-references, the index, or the table of contents for moving quickly from one point in the document to another. The table of contents may also be used for editing the document in outline form.

The components of a structured document representation

The structured document model representation can be viewed as being object-based, with higher-level objects formed by composing more primitive objects. In describing the structured document representation in more detail, it is useful to make an informal distinction between the representation's *primary structure* and its *secondary structures*. The primary structure is the predominating structure; the relationships that define the form of composition of primitive objects into higher-level objects. The primary structure described in this paper is tree-based. The secondary structures represent additional relationships among the document's objects; relationships that do not directly affect the composition of higher-level document objects. Common secondary relationships are those expressed by cross-references, and those expressed by the relationship between a floating object (such as a figure or perhaps a footnote) and the reference to that object.

More precisely, the primary structure used here is a forest of trees. Some elements, figures and notes for example, may be put in different places in a document when the document is formatted. A note may be laid-out at the bottom of the page where it is called for the first time, or at the end of the chapter, or even at the end of the book, with all other notes (this is often the case for bibliographic notes). A note may be called from several points in a document. Such elements do not have a fixed logical position in the document tree and can be related to several nodes of the tree. Consequently, such elements are represented as a *separate* tree from the main body of the document and the separate trees in the forest are related to one another with secondary relationships.

A further issue in the definition of the components that make up a structured document is the identity of the atomic object or objects—the most primitive objects. Two relevant choices to be made in design are the size of the object represented as atomic and whether there is a single class of such objects or multiple classes. In this paper's document model, the atomic objects are relatively large in size and heterogeneous in form. For example, the atomic object is perhaps a string of text (and not the individual character), is perhaps a line drawing (and not an individual element of that line drawing), and so on. The

reason for this is to encapsulate non-tree structures (or variant tree structures) into the atomic objects, thereby permitting the primary structure to be the relatively simple, well understood, and easy to manipulate tree structure.[4]

The encapsulation of structures into an atomic object may be for one of two reasons. In some cases, such as strings of text, the tree structure is more complex than the most natural representation for the object. In other cases, such as tabular material, which is most naturally represented as a matrix, the natural representation does not translate straightforwardly into a tree. Similarly, a bitmap is not straightforwardly represented as a tree. Variant tree structures, differing from those used to represent the primary structure and encapsulated within a leaf of the general tree structure, may be convenient to use for mathematical equations or line drawings.[5] WYSIWYG-like systems have also incorporated such collections of atomic objects, imbedding them in a linear structure rather than in the tree structure we use for structured documents; Xerox's Viewpoint[13,14], Interleaf's Publishing System[15], and the Apple Macintosh's MacWrite[16] are examples.

Grammatical definition of structure

The structured document representation (indeed, any document model representation) may be considered at two different levels: specific and generic. The distinction may be seen when considering document classes. A document class serves to group documents with similar structures. Examples of classes include letters, reports, manuals, and books. The generic logical structure represents the model for all documents in a given class. The specific logical structure represents an instance of a document in a class.

The atomic objects defined by a system are likely to be common across all classes of documents. In other words, the same basic components are likely to be found in books, letters, and so on. What will distinguish one class of documents from another is their respective primary structures—the identity of the higher-level objects and the manner in which they are composed.

An issue is whether the object interrelationships should be constrained or unconstrained. If the interrelationships are unconstrained, any set of objects may be composed into a higher-level object. Such a design is attractive because it eliminates the need to verify the correctness of the composition.

Our systems, however, incorporate a grammatical specification of the object interrelationships in the generic logical document. An expense of this design decision is that it becomes more complicated to design a simple to use but powerful user interface because of the provisions that must be made to allow the system's user to determine what object relationships are permissible during specification of the document. Indeed, it is sometimes useful (or even required) to guide the system's user through the creation of the structure, thereby ensuring that the constraints on relationships are maintained. However, the added complexity of system design is offset by the increased flexibility in commands that can be based on information gained from the grammatical specification.

The key point is that a grammatical specification allows the system to incorporate an

[4] It is relevant to emphasize that if such structures were not encapsulated into the leaf objects, the primary structure would be required to be rich enough to permit representation of all desired relationships, perhaps a rooted directed acyclic graph structure.

[5] Grif does not define variant tree structures, representing such objects with the standard tree. The **pedtnt**, however, does use a variant tree to represent mathematical material.

'understanding' of the components of the structure. The grammatically-defined structure allows the designer of the document class to specify whether or not a particular object may be contained within another. Additionally, the designer can specify that certain document elements are to be optional and others are to be required. A system can enforce this aspect of the design, insuring that required elements are provided by the author.

Since a grammatically-based specification provides a convenient and well-defined framework, extra-grammatical mechanisms also can take advantage of this framework. As noted before, an advantage of the structured document representation is that system-computed information may be associated with elements of the document—perhaps numbers or other identifying information. The specifications of how these values are to be computed also may be associated with the grammar elements, thereby allowing the flexible definition and modification of these facilities as well. Finally, as we will discuss later, the elements in the grammatically-based specification provide a base for attaching the information that guides the translation from logical structure to physical appearance.

A special case of these mechanisms is that which permits special representations of the document—perhaps showing only a selected subset of the document's objects, or perhaps presenting the same objects in differing representations.

Aspects of a user interface

WYSIWYG editor/formatters present the interactive representation of the document as identical to the printed representation. In other words, the form appearing on the screen has all the fonts, sizes, and spacings of the form that is printed on paper. Editing changes are performed directly on this representation of the output. The resulting user interface is natural-seeming as the display of the document and the manipulations that take place are similar to those that would be carried out for paper documents. Such naturalness is a goal for the user interface of any interactive system.

It is important to distinguish between the goals of systems such as those described in this paper and those of the WYSIWYG editor/formatters, as such distinctions affect the characteristics of the user interface. Indeed, the WYSIWYG user interface, while natural in form, limits the document representations that can be used. The characteristics of a WYSIWYG user-interface can be further subdivided into three categories:

1. *Direct manipulation* style of interaction[17]: the user's operations are seemingly performed directly on the displayed representation and are expressed in terms of that representation, perhaps through a pointing device such as the mouse.
2. *Exact-representation* of the printed output: the displayed representation is the same as the printed representation.
3. *Immediate recognizability* of document components from their appearance on the display.

It is the exact-representation characteristic that is most in conflict with a structured document representation.

Exact-representation restricts the power of the algorithms that may be incorporated into a system. The need in an exact-representation system is to use algorithms whose actions over the document are as limited and as predictable as possible. In part this is a technology-imposed limit—such algorithms must be fast and must operate incrementally in order to permit the editor/formatter to be responsive to the user's commands. But

in addition there are psychological advantages to limiting the scope of change in a document to an area that is closely associated with the point of change. This restriction conflicts with the need of the algorithms that are intended to produce high-quality output representations, for example those that have been associated with TEX. These algorithms attempt to examine as *much* of the document as possible in determining how to display the document's objects, and changes to the display may range over wide areas of the document, and are not necessarily directly associated with the point of change.

Our systems' user interfaces relax the requirement for exact-representation but retain a direct manipulation interaction style and provide immediate object recognizability when appropriate. The display shown to the system's user is not intended to reflect the final form of the document.[6] Indeed, there is no single exact-representation of a structured document—for example, many different layouts may be generated from the same logical structure. Thus when editing a structured document, the displayed representation is intended primarily to reflect the logical structure of the document and to permit the system's user to see the results of commands as they are entered. Such representations can be specialized to the editing task—object boundaries may be represented explicitly, pagination may not be represented, material located in separate trees of the structure may be displayed in separate windows, etc. When printing, the system may use sophisticated algorithms for setting the text into lines with hyphenation, kerning and ligatures or for building pages. Such processing can be done separately from the editor and consequently the interactive performance of the algorithms used is not critical.

A further issue in the design of the user interface is the mechanisms provided to allow use of a variety of leaf objects. As mentioned previously, such objects can range widely both in the components they are built from and also in the structure they represent. For example, components may range from textual material to line drawing-oriented material. Structures may range from linear structures, as in a string of text, to a two-dimensional matrix, as in a table, and to a tree structure, as in a mathematical equation. Clearly the design of the enclosing structure interacts with the design of the leaf objects' structure(s), and with the components included in the leaf objects. If separate editors are associated with each of the kinds of leaf objects, it is necessary to design carefully to insure that the resulting user interfaces are compatible with each other. Similarly, it is important that the commands provided to edit the leaf objects be compatible with the commands that edit the structure, itself. A question that we will explore further later in this paper is the degree to which the boundaries between the structure editor and the leaf object editors can be removed in the user interface design (although the separate editors would still exist in the underlying implementation).

In understanding the issues in design of user interfaces that permit direct manipulation of structured documents, it is worth considering the similarities and differences between documents and computer programs. Programs are documents of a particular type. Like the structured documents considered here, programs are highly structured and grammatically defined, and therefore specific tools have been created for manipulating them efficiently—for example, programming environments have incorporated syntax-directed editors. We must consider why we do not simply apply these well-understood syntax-directed editors to documents in general.

[6] Grif allows definition of a display that resembles the final form, but this feature is usually used only with a completed document, just for checking before printing.

Programs are a very special class of documents. Programs are defined syntactically and semantically with a great deal more precision and rigidity than are printed documents. On the other hand, printed documents are typographically much more complex than are programs.

The structure relating programming language constructs is homogeneous, and tree-structured but we find it convenient to define heterogeneous structures to describe documents. There does not seem to be the same need in the programming language domain for the variety of object relationships commonly found in the document domain.

It is perhaps not surprising that tools, such as syntax-directed editors, with a strong orientation to the grammatically defined object relationships would more naturally manipulate programs than documents. Grammatical specifications have been applied to documents primarily as a means of *describing* the relationships of their components. They serve as the means of *defining* those relationships in programming languages.

Converting from logical structure to physical appearance

A critical component of a document processing system, particularly one that is based around structured documents, is the means by which the mapping to a displayed form is specified (either paper or CRT). In the context of structured documents, such specifications often are associated with the individual grammar elements, indicating in some fashion how the associated objects are to be displayed. One possibility is to associate physically-oriented attributes with the elements (for example, changes to the margins, vertical spacing separating the associated object from its neighbors, etc.), where the values specify changes to global variables. Another is to associate action routines with the elements, computationally specifying changes to the global variables.

A different approach is to specify the document's layout explicitly using a separate language (perhaps expressed as an interactive tool). This technique more closely resembles those physical layout techniques in which a page template is prepared to indicate where material is to be placed.

Specifying the physical appearance of the elements of the document is not the only aspect of the mapping from the logical representation. The mapping may not be one-to-one—certain objects may be suppressed on the display or other elements may be added to the display. As an example of object suppression, consider the formation of a table of contents in which only the section headers are retained. As a second example, Grif permits filtering of the objects in the logical representation, allowing specification of parallel versions of a document in different languages. The mapping to the display may show perhaps only one of the versions or it may present them all. As an example of object generation, consider the addition of header material to particular objects, for example the string 'Abstract' before the text of a document's abstract.

Interactive manipulation of structured documents

We will now turn our attention to the specific features implemented in Grif and in the `pedtnt`. The two systems share similar document models. To begin the discussion on the systems' specifics, we will first describe the two models, discussing the motivation for the differences that do exist between the two. We will also inspect Grif's specification of the physical appearance of components in the document's logical structure—in other

words, the specification of the mapping from document model representation to output model representation. Finally, we will turn our attention to user interface issues. Grif's user interface is more highly developed than is the **pedtnt**'s and provides a good illustration of what can be presented to the user. The **pedtnt**, on the other hand, has served as a testbed for techniques that allow the direct manipulation user interface paradigm to be applied to the structure of the document and not limited to acting within the bounds of a particular document object. We will, therefore, concentrate on describing Grif's user interface and then turn to a discussion of the **pedtnt**'s extensions and their relevance to such interfaces.

Definition of document structure

The structures incorporated by Grif and by the **pedtnt** are based on a forest of ordered trees, where each tree represents a different *stream* of the document, as discussed earlier.

Each node in the tree is associated with a document object, and the relationship between the objects is specified grammatically (these specifications will be discussed later in this section). The objects located at the leaves of the ordered tree are relatively large in size, for example strings of text. In both designs, the leaf objects are heterogeneous in form. Grif defines four types of leaves: text (a character string, in a given alphabet, for example latin, greek, or cyrillic), picture (a rectangular bit-map, one bit per pixel), graphic (a geometric shape, for example a vector, a rectangle, or a circle), and math symbol (a character that can be stretched such as a surd, '$\surd$', an integral symbol, parentheses, braces, and arrows).

In addition to the tree relationships, Grif defines *references*. A reference is a bidirectional link between two nodes of a same tree or of two different trees. One of these two nodes is a node of a special type (reference) and is a leaf in its tree. These reference nodes are used for representing cross references. The link is bidirectional because it can be used to find the referenced node from the reference and from a referenced node to find all its references.

The Grif tree structure is used to represent the document's structure, to represent mathematical equations, and as an interim measure to represent tabular material. The implemented representation of equations is **eqn**-like in structure—representing the interrelationships of the graphical elements but not necessarily the mathematical meaning of the equation.[7] The equation's structure and displayed representation are specified using the same mechanisms that are used to specify the structure of the document in general. The table representation is either row-major or column-major in the current design, although extensions to the semantics of the references are being considered to more closely model the two-dimensional matrix structure of tables.

The relationships among Grif's document objects are described using a 'structure schema', which resembles a Pascal record in syntax. This structure schema used to describe the document's generic structure specifies sequences of document objects (delimited by the **begin** and **end** keywords), where each object may itself be a simple object, a document object repeated one or more times (specified with the **list of** keyword), or a selection of one of a collection of document objects (specified with the **case of**

[7] This statement is actually an oversimplification. Since Grif's equations are defined by structure schemas, alternate representations could be defined that more closely resembled the equation's mathematical meaning.

keyword). Grif's display of the document is driven by a 'presentation model', which provides a description of how the process of displaying a particular object is to modify the values associated with global appearance-related attributes (e.g., the width of the display area, etc.). In other words, the presentation model defines the spatial relationship between a box representing the document object that is being displayed and the enclosing and adjacent boxes representing that object's parent and siblings in the tree that represents the document. The presentation model also resembles the Pascal record and will be discussed in more detail later. Further details also will be given of Grif's user interface and of the syntax of the structure schema later in the paper.

The grammar must provide constructs for flexibility. It must allow the user to choose the type of certain elements either among a limited list of types or without any limit. But flexibility may be improved in the editor itself, according to the way it uses the grammar. Grif considers that the generic logical structure described by the grammar is to used to *suggest* to the user the type of the elements that are to be created in each context, not to *require* the creation of those elements. Thus each of the grammar-specified elements may be treated as optional by the system's user. When the grammar specifies that a report contains an abstract, the editor generates an abstract automatically as soon as the user creates a report, but it also allows the user to delete this abstract at any time. However, if a document must strictly conform to its grammar, a command allows the user to generate all elements defined by the grammar for a selected part of (or the whole) document.

As in Grif's logical document representation, the **tnt** is based on an ordered tree (called the *strict tree*). Three types of leaves have been defined in the **pedtnt**: the text-block (a string of text), the table-block (a two-dimensional matrix representing the structure of the table), and the equation-block (a tree, but of a different form than the strict tree that encloses the equation). As may be seen from this list, the leaves themselves possess a structure and are defined to terminate either in *atoms* or in *transition nodes* or in both. An atom is an actual terminating point in the **tnt**. The transition node serves as the root of another instance of the enclosing strict tree. This distinction is made primarily for tables—a new instance of the strict tree is associated with each of the entries in a table, thereby allowing specification of contents. A special form of character that stretches to fill the given space is provided in the equation-blocks—this corresponds to the Grif math symbol but is encapsulated within the **tnt** equation-block and not a separate kind of leaf.

The **pedtnt** is a prototype system and implements a subset of the document model, manipulating a single **tnt**. The document model, however, represents the document as a forest of **tnt**s, associated with one another through *links*, which resemble the Grif references in intent. Links originate in a contiguous range of leaf nodes in some **tnt**, and terminate at the root of another **tnt**. Any number of *visible reference points* may be associated with a link and placed within the origination range of the link. These points serve as atoms within the **tnt** and are meant to expand to be a string that identifies the target **tnt** when the originating **tnt** is displayed.

The **pedtnt** user interface is driven by specifications written in a context-free grammar whose syntax is slightly modified from the BNF. The grammar's syntax has been augmented by addition of operators that signify selection of components, and that signify repetition of the component. Restrictions on the form of the grammar have also been included to simplify **pedtnt**'s processing. In addition, the terminal symbols have been further separated into user-defined terminals (whose value is to be provided by

the user) and system-defined terminals (whose value is to be computed by the system). Unlike Grif's specification, elements identified in the grammar *must* be included in the document and cannot be omitted. The form of the grammar is currently being modified to permit specification of optional material.

The primary difference in design between Grif's logical representation and the **tnt** is the presence of variant tree structures in the **tnt**'s equation-block. The **pedtnt** incorporates a separate editor for each type of leaf node, specialized for the object represented by that leaf node. Grif, on the other hand, tries to use a homogeneous representation whenever possible, using its standard tree and tree manipulation commands for mathematical material. The trade-off is that the **pedtnt** approach permits incorporation of manipulation operations specialized for a particular leaf object but requires development and maintenance of a separate editor for the object.

Mapping from logical structure to physical appearance

Thus far the emphasis has been put on the logical structure, but when a document is printed or displayed, a layout (or graphical) structure is needed. In this section, we will consider the mechanisms provided by Grif for describing this layout structure and for specifying the translation from logical structure to layout structure[18].

Like the logical structure, the layout structure may be considered at two different levels. There is a generic level, which represents the model for all documents in a given class, and a specific level, which represents an instance of a document in a class.

At the generic level, a set of rules, called presentation rules, define the layout and the physical appearance of all types of elements specified in the logical generic structure. For the class 'report', for instance, these rules indicate that the title must be set in large characters, centered in the page and placed at one inch from the top. Rules may also indicate that paragraphs are set with the body font and size defined in the surrounding environment: in other words, that they inherit some presentation parameters from their parent in the logical structure. Thus, if presentation rules state that a footnote is displayed in eight point size type and that the content of a section is displayed in eleven point type, paragraphs in a footnote are displayed in smaller characters than paragraphs in the main text, even if the same type of paragraph is used in both cases.

In order to achieve the integration of all document components, the same kind of presentation rules should be associated with textual elements (for example, paragraphs, headings, and title) and with other types of elements (such as a cell or a row in a table, or an exponent or a denominator in a mathematical formula). It is then not possible to use traditional typographical attributes in presentation rules, as the notions of margins or line spacing or indentation have no meaning within a formula or a table. In Grif a general presentation model has been defined. It allows the user to describe in a uniform way the layout of many types of components: text as well as formulæ, tables or graphics. It is based on the concept of a *box*. The box is the (normally invisible) rectangle that delimits an element on the screen or on the paper. A box is associated with each element in the specific logical structure of a document, and presentation rules define how this box must be built by the editor.

The presentation rules define the box's position, its dimensions, and the way to display its content. Position and dimensions may be defined relatively to the other boxes—for example, the box associated with the logical element 'title' may have a set of presentation

rules for specifying its width (for example 80% of the width of its parent) its horizontal position (centered within the box of its parent) and its vertical position (one inch from the top of the box of its parent). The same kind of rules may be used for defining the position of a numerator within a fraction or a cell within the column of a table.

Other rules define the way to display the contents of a box: the font and the layout of the content. There are rules for specifying the font and its attributes (e.g., font family, size, boldness, and italics) and these attributes may be inherited. There are other rules for specifying the way to lay out the elements contained within the box. These elements may be placed according to their own rules (like the title within the report, or the denominator within the fraction), or according to a rule associated with the box of their parent. This latter case is used for example for paragraphs, where characters and structured elements (formulæ for example) are placed each to the right of the previous one until the right limit of the box is reached (a new line will be created when the limit is reached). The display of lines in the environment is defined by additional rules, for example rules specifying the inter-line spacing, the indentation of the first line in the element (as in a paragraph), and the relative indentation of the remaining lines in the element.

Presentation rules are also used to generate new elements, called presentation elements, according to the logical structure of the document. These rules are used for generating numbers (e.g., section number, chapter number, formula number, or note number) or for adding short text strings, like ‘Abstract’ or ‘Appendix’, which make the logical structure more evident to the reader. Instead of short text it is also possible to add graphic elements: the bar within a fraction (a horizontal line), a box drawn around a cell in a table, or a hair-line after a title, for example.

Presentation rules are grouped into ‘presentation models’. The Grif editor needs a presentation model to display a document. A presentation model is associated with a document class and defines the way to display all the types of elements defined in that class. Several presentation models may be associated with the same class, allowing the documents to be shown in different ways. The user may choose a presentation model according to his/her taste or to the work he/she wants to carry out on the document. When creating a new document from scratch, a presentation model that shows the structure explicitly is useful. Such a presentation model may augment the display with additional information, for example the type of the elements it creates, in order to better help the user’s task. On the contrary, when reading an existing document, it may be easier to use a presentation model that shows the document as it would appear if printed. It is important to note that the system’s user may switch between the defined presentation models at will (or may use several simultaneously). It is possible to use the editor as a WYSIWYG system, by defining an appropriate presentation model that is used both for editing and printing. But when structural modifications are being made, another presentation model may be chosen.

In each presentation model it is possible to define several views that will be displayed during the editing of different aspects of the document. In a presentation model, there are rules for defining which elements will be visible in each view. Visibility may depend on the type of the element and/or on its hierarchical level. Thus it is possible to specify a view (called table of contents) that displays all the section headings that are located above a certain level, suppressing the rest of the document. It is also possible to specify a view that displays all mathematical formulæ, whatever their level in the document structure. For a class of bilingual documents one can define a global view displaying all

elements, an English view and a foreign view, each of which displays only elements in one language.

When editing, the user is free to open or close any view defined in the presentation model in use. The user is also able to modify some of the presentation parameters for a given element, changing the specific presentation of the element, for example, its position, dimension, font, body size, justification, line spacing, and so on. Each view is displayed in a different window and may be edited in a uniform way. It is, for instance, possible to create an outline of a new document just by using the table of contents view, and then to type the text in the global view. Even then, it is possible to go back to the table of contents for creating a new section, or moving to another part of the document or permute two sections. Any action in one view is immediately reflected in the other existing views where it is visible.

THE GRIF USER INTERFACE

We have presented the general mechanisms that Grif uses in specifying the structure of different classes of documents (the structure schema) and the mapping from that logical representation of the document to a physical presentation of the document (the presentation model). In this section, we will examine the ways in which these specifications are incorporated into an interactive system. We will examine the Grif user interface through a series of examples.

Figure 1 shows two different documents belonging to the same class and edited with the same presentation model. For each document only one view is open, the Global view. Window B (at the bottom left corner) displays an empty document, just after the user has created it. The editor has generated the elements defined by the generic logical structure, which specifies that the body and the appendices contain at least two sections. Of course, the user is free to delete any element created by the editor. Window A (above and to the right of Window B) contains a second document that has been developed from the starting point shown in Window B.

The grey rectangles in the windows represent empty elements. In Window B, the user has selected with the mouse the first (and presently only) paragraph of section 1, which is now displayed in dark grey. He/she has then clicked the first entry of the permanent menu, located at the top of the screen (‘Create Within’ the selected element), and the system displays a pop-up menu for selecting the type of the element to be created as a paragraph. This menu is built following the generic logical structure.

The permanent menu contains eight entries—the last four entries are constant and the first four entries depend on the current selection, allowing the creation of new elements. When the current selection is empty, new elements can be created within the selection (as in this example). In any case, elements can also follow the current selection and the permanent menu indicates that such elements can be created at a number of different levels (in the example, a paragraph at the same level, a sequence of subsections at the next upper level, or a section at the second upper level). Creation commands are flagged with the ‘*’ in the menu.

Window A displays a document that has already been edited. The user has deleted the ‘Affiliation’ element and has created an additional author. The abstract is not yet written, but the heading and some of the contents of section 1 have been specified.

This presentation model has been designed for creating a new document or for updating

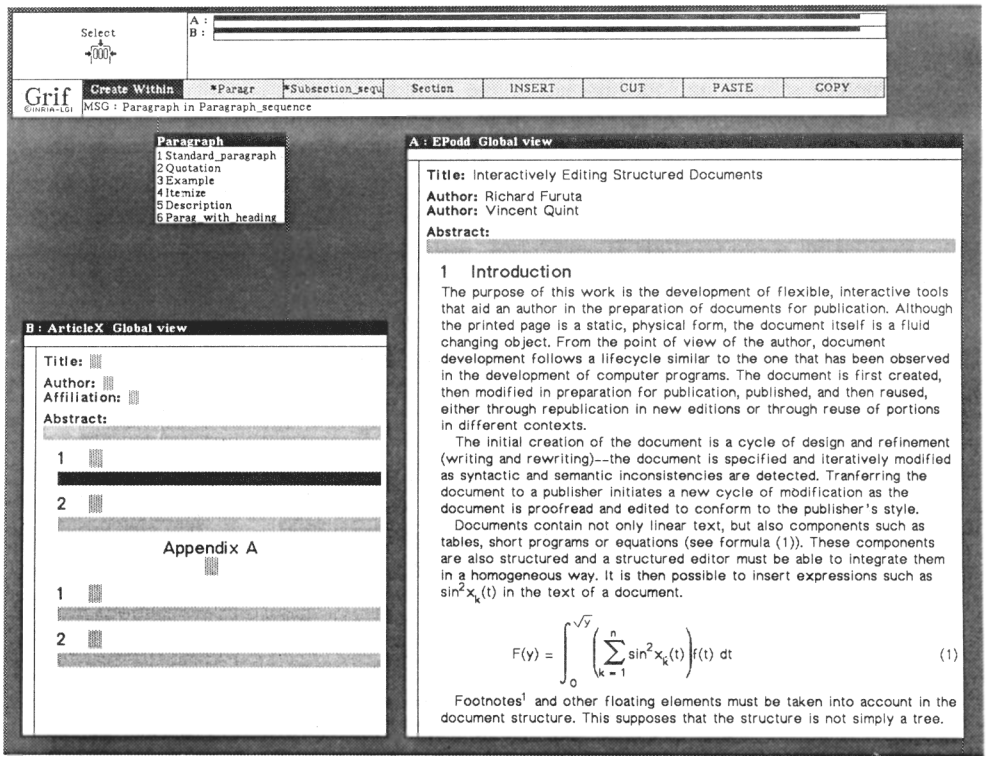

Figure 1. The global view of two different documents

an existing one. Most elements are introduced by a short text string indicating their type, there is no pagination, no justification and the text is printed in a large, readable font. The line length is not fixed, and depends on the window width.

In Figure 2, Window B has been closed, and two additional views have been opened for the document of window A (‘Table of contents’ and ‘Notes’). As the presentation model is not paginated, ‘footnotes’ are shown in a different view, Window C in this figure.

The user has selected the heading of first section, by clicking it with the mouse. It is now displayed in reverse video in all views where it is visible. The type of the selected element(s) is displayed in the MSG area of the Grif dialogue window, on top of the screen.

As some elements already exist after the selected element, the variable part of the permanent menu allows selection of the next elements at different levels (first paragraph of introduction, next section, or the appendixes). This allows the user to move quickly across the document. The views are another way to move across the document. Clicking in the table of contents selects a new element, which is then displayed in all views where it is visible—the Global view will display the beginning of the selected section.

The user wants to create a new element and has clicked the entry INSERT in the permanent menu. As a result a pop-up menu is displayed allowing choice of the type and

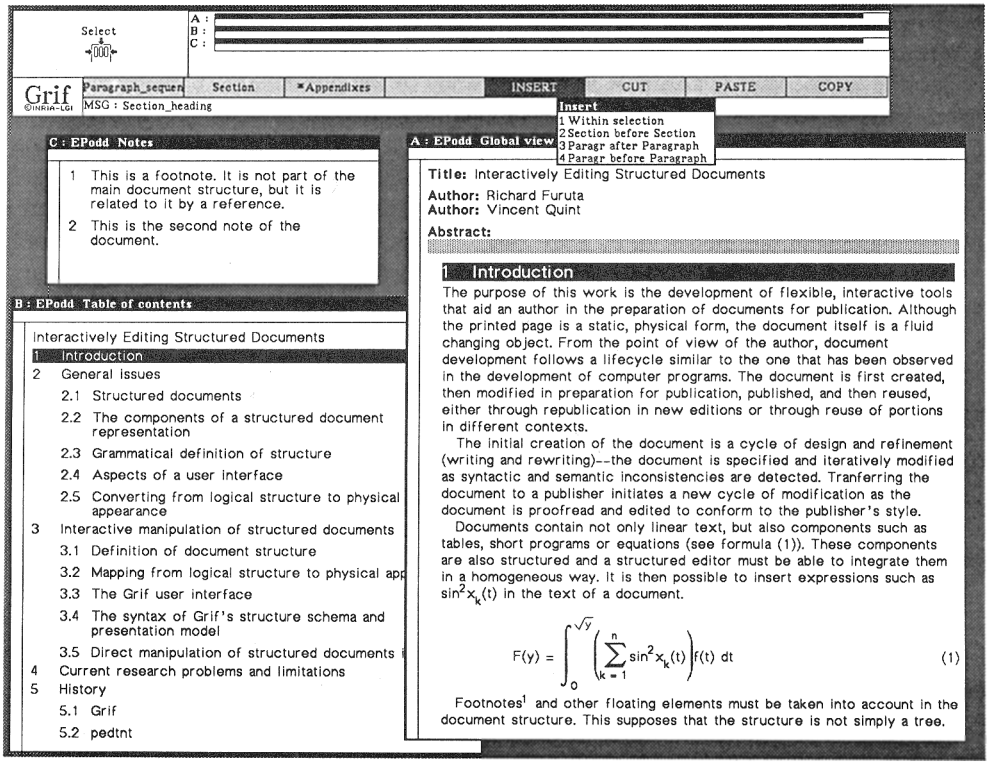

Figure 2. Three views of the document

position of the element that is to be created. The choices here are to create a new section before the introduction, a new paragraph after the (empty) paragraph of the abstract or a new paragraph before the first paragraph of the introduction.

The constant entries of the permanent menu are used for activating the generic commands (Insert, Cut, Paste, Copy), which act on the selected part of the document, whether it is a structured part or not. Moving (Cut and Paste) a character string is performed exactly in the same manner as moving a section, for example.

In Figure 3, the user has closed the Table of contents and Notes windows and has changed the size of Window A. The layout has been automatically modified, taking into account the new window size, as specified in the presentation model. The user has then created a table and put some structured and unstructured elements in the cells. The cursor is now at the end of the equation in the table and the user has clicked the first entry in the permanent menu, for creating a new mathematical Construct in the equation. The pop-up menu displays all possible constructs, as specified in the generic logical structure for formulæ (Figure 5 will give more details on mathematical formulæ specification). The user may then choose the construct to be appended to the formula.

Although the screen contains a number of (permanent and pop-up) menus, the user is free to use the mouse and click on the screen, or to use the keyboard. There is a function key associated with each entry of the permanent and pop-up menus. The numbers in the pop-up menus represent the number of the function key associated with each entry.

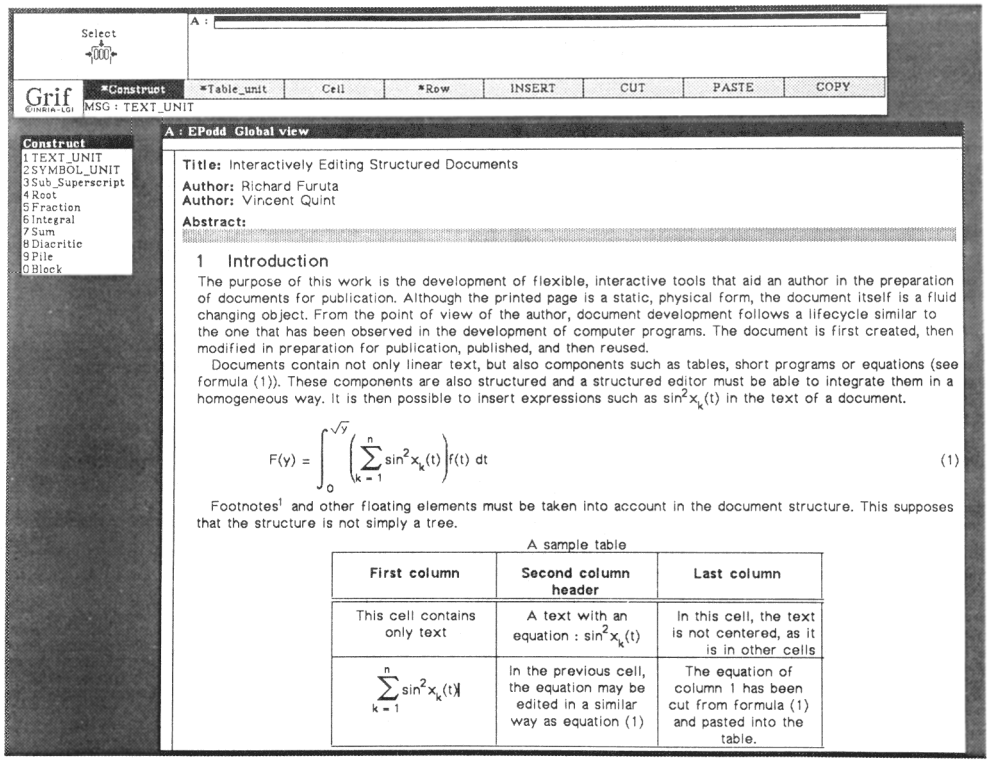

Figure 3. Continuing the specification of the document

Finally, in Figure 4 the user continues to work on the same document, but has invoked a different presentation model, which shows the final output as it would be printed on a laser printer. Only the presentation model has been changed—none of the document specification has been altered. This presentation model specifies pages of fixed width and height. The presentation model also specifies that notes are displayed as page footers, and not in a different view. Window A displays the first page of the document.

Two additional windows have been opened, one for the Table of contents view, and one for the Formulæ view. The formula may be edited in either the Formulæ view or in the Global view. Switching between views is carried out by moving the mouse. Similarly, the title and section headings can be edited in the Table of contents view or in the Global view. If the user exchanges the headings of two sections in the Table of contents view, the corresponding section body will be moved as well.

A Search menu has also been selected. The system can search for text, like an ordinary text editor, for mathematical symbols (for example the integral symbol, the root symbol, or the sum symbol that appear in the formula), for graphic elements (line, rectangle, circle, etc.), for elements of a given type, for elements having a given logical attribute, for all cross-references that refer to the selected element, or, if a reference is selected, for the element that it points at. For example, if equation (1) is selected (in total or in part), clicking the Reference entry of the Search menu will select all references to that equation, one after the other (for example, there is a reference to the equation in the paragraph that precedes it).

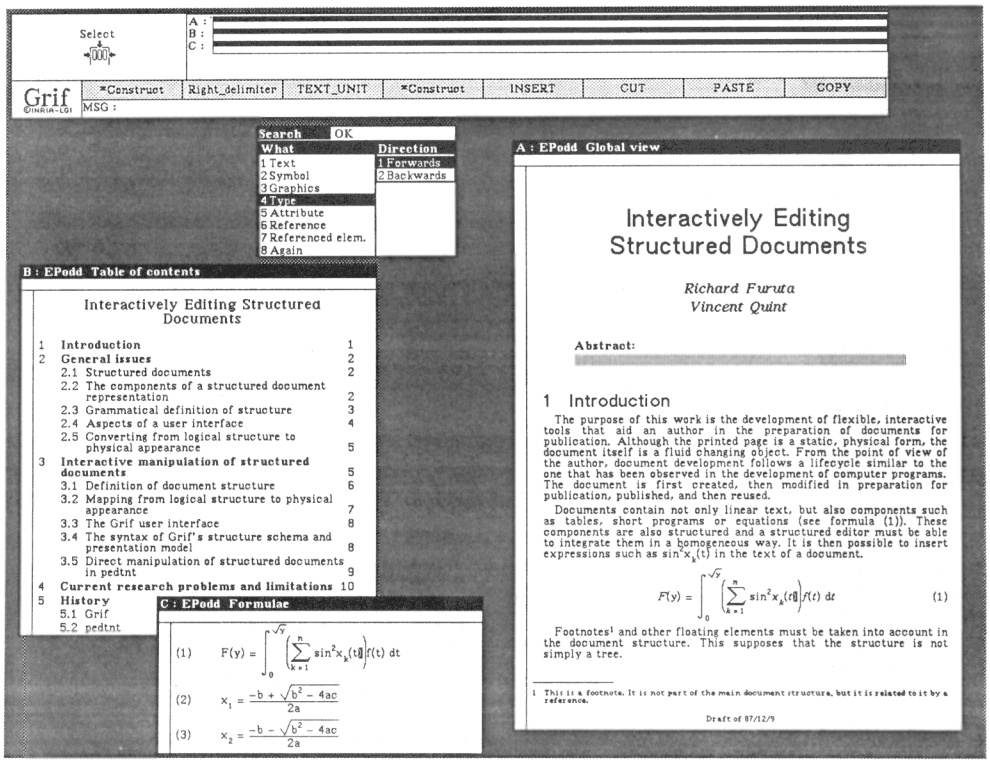

Figure 4. Document presentation resembling printed output, and additional views

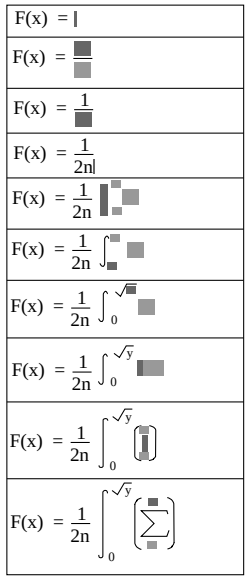

Figure 5. Editing mathematical formulæ

Figure 5 shows the steps in editing a formula in Grif. As the formula is described using Grif's tree, the generic logical structure used to define the formula is described in the structure schema and the display of the formula is specified by the presentation model. In this presentation model, creation of an empty formula results in a small grey rectangle. The user has typed '`F(x) =`', resulting in the top picture in the figure. Next, a fraction is created, resulting in the next picture. The dark rectangle indicates where typed characters will be placed. The user then types '`1`', clicks the 'Denominator' entry in the permanent menu (picture 3), and types '`2n`' (picture 4). Specification continues in a similar fashion until the complete formula is displayed.

The syntax of Grif's structure schema and presentation model

Having seen Grif's user interface, let us return briefly to the Pascal record-like syntax of the structure schema and of the presentation model. Figure 6 shows a document of a different class from the earlier examples, called a bilingual document. Figure 7 shows the complete structure schema that corresponds to this class. Note that parallel versions of the structure's leaf objects are specified—one flagged as being in the English language and the other in the French language. The document's author would enter both versions.

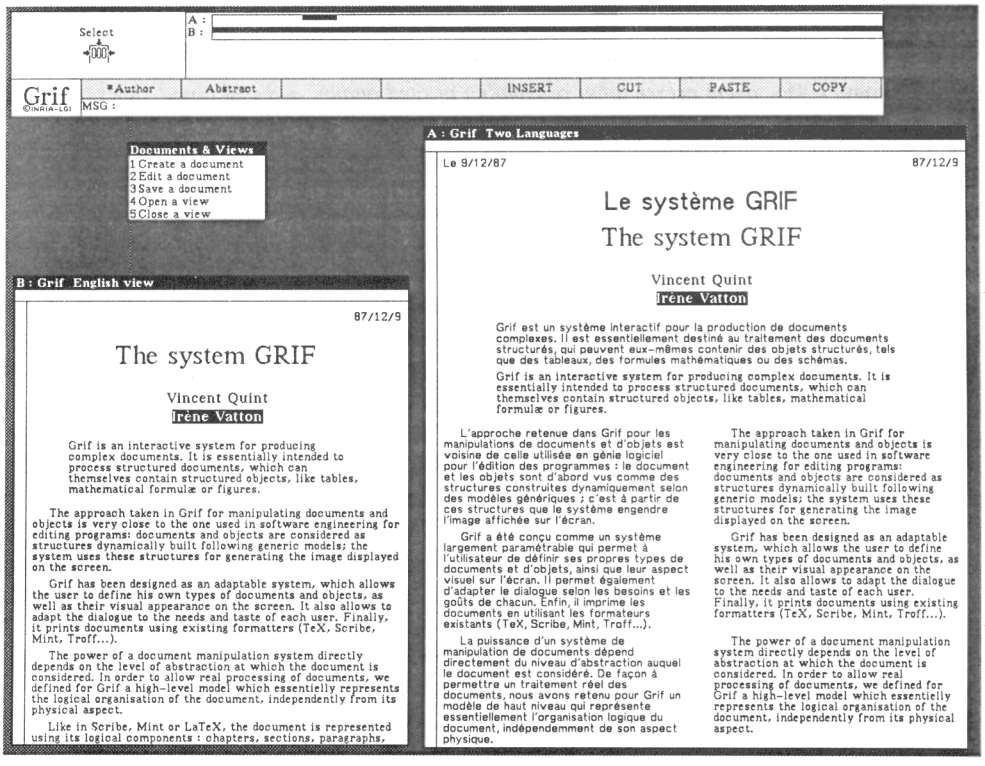

Figure 6. A bilingual document

```
STRUCTURE Bilingual;    { class name }

DEFPRES BilingualP;     { default presentation model }

ATTR                    { logical attributes definition }
   Language = (French, English);

STRUCT                  { logical structure definition }
   Bilingual =
        BEGIN
        Title = BEGIN
                French_title = TEXT  WITH Language=French;
                English_title = TEXT WITH Language=English;
                END;
        Authors = LIST OF (Author=TEXT);
        Abstract = BEGIN
                   French_abstract = TEXT WITH Language=French;
                   English_abstract = TEXT WITH Language=English;
                   END;
        Body = LIST OF (Bilingual_parag =
                        BEGIN
                        French_parag = TEXT WITH Language=French;
                        English_parag = TEXT WITH Language=English;
                        END);
        END;
END
```

Figure 7. The structure schema for the bilingual document class

```
DEFAULT { default presentation rules. These rules are applied to all
          elements, except when the corresponding rule is specified for
          the type of the element }
   BEGIN
   HorizRef : * . Top;                  { horizontal base line of the box }
   VertRef : * . Left;                  { vertical base line of the box }
   Width : Enclosing . Width;           { box width }
   Height : Enclosed . Height;          { box height }
   VertPos : Top = Previous . Bottom;   { vertical position of the box }
   HorizPos : Left = Previous . Left;   { horizontal position of the box }
   Justify : Enclosing =;               { justification of the contents }
   LineSpacing : Enclosing =;           { line spacing of the contents }
   Break: Yes;                          { page break allowed within box }
   Visibility: Enclosing =;             { box visibility level }
   Font : Enclosing =;                  { font family: Times, Helvetica... }
   Style : Enclosing =;                 { font style: roman, bold, italics }
   Size : Enclosing =;                  { character size}
   Indent : Enclosing =;                { first line indent in paragraphs }
   END;
```

Figure 8. Default presentation rules for the bilingual paragraph

```
RULES   { presentation rules for the types defined in the generic structure }
   Bilingual_parag :
        BEGIN
        VertPos : Top = Previous . Bottom + 0.5;
        HorizPos ; Left = Enclosing . Left;
        Break : No;
        END;

   French_parag :
        BEGIN
        Width : Enclosing . Width * 48%;
        VertPos : Top = Enclosing . Left;
        HorizPos : Left = Enclosing . Left;
        Line (Left);
        IN French_view
           Width : Enclosing . Width;
        END;

   English_parag :
        BEGIN
        Width : Enclosing . Width * 48%;
        VertPos : Top Encloding . Top;
        HorizPos : Right = Enclosing . Right;
        Line (Left);
        IN English_view
           Width : Enclosing . Width;
        END;
```

Figure 9. The bilingual paragraph

```
ATTRIBUTES      { presentation rules for logical attributes }

   Language = English :
        BEGIN
        Font: Times;
        IN French_view
           Visibility: 0;
        END;

   Language = French :
        BEGIN
        Font: Helvetica;
        IN English_view
           Visibility: 0;
        END;
```

Figure 10. Attributes associated with the structure schema

The presentation model defines three different views: one bilingual, one English, and one French. Only two have been opened here. The pop-up menu shows the commands for managing documents and views.

Figures 8, 9, and 10 show portions of the presentation model representation. The definition begins with association of default presentation rules with the elements of the box, shown in Figure 8. Specific rules are then associated with each of the presentation elements defined in the structure schema. The fragment shown in Figure 9 gives the rules for the box corresponding to the `Bilingual_parag` and of its two components: the `French_parag` and the `English_parag` (unsigned scaling factors are expressed in terms of the current character size). The `Line` entry specifies how lines of text are to be justified. Note that the `French_parag` and `English_parag` contain conditional definitions, which are related to the particular view that has been selected. Finally, Figure 10 shows presentation rules that are associated with the `ATTR` specifications in the structure schema. In particular, these attributes select the display font for the French and English segments of the text and suppress display of one language's specification when the other language's view has been selected.

Direct manipulation of structured documents in `pedtnt`

It is important to note that while the representation of the document is structured, the user's view of manipulations over the document can, in many cases, be quite similar to those presented by the WYSIWYG editor/formatters. An important class of commands that can be presented in such a way are those that operate over selected regions of the displayed text. In such operations, a cursor is initially positioned at one end or the other of the selection and cursor motions cause the selection to expand or contract. An action is applied to the selected region. Standardly, WYSIWYG editors permit their users to delete such a selected region, duplicate the region, move the region (semantically, duplication and deletion merged), or change some attributes of the display within the region (perhaps the font of the display).

Analysis of the visual effects of these region-oriented WYSIWYG commands has shown that these effects can be duplicated in the context of the **`pedtnt`**. Unlike the WYSIWYG document representation, operations applied in the context of the structured document must maintain the correctness of the structure itself. When an operation such as 'delete' or 'move' modifies the document's structure, the effects on the structure are defined through heuristically-defined algorithms. Not all operations affect the structure—actions such as 'change font' affect only the attributes associated with leaf nodes. Such actions can be carried out by isolated application to the appropriate nodes.

Operations that modify the structure are permitted to leave 'holes' in the **`tnt`**—in other words they are permitted to omit document elements that are required by the grammatical definition. They are not, however, permitted to produce object orderings that cannot be matched to the grammatical definition. These operations proceed in three steps:

1. The region is marked by the system's user and the action selected.
2. The action is carried out, potentially leaving out parts of the **`tnt`**, as described.
3. The **`tnt`**'s correctness is restored by matching the elements that are present to those specified in the grammar, creating new instances of any missing elements.

It is the second step that potentially modifies the structure and the point at which

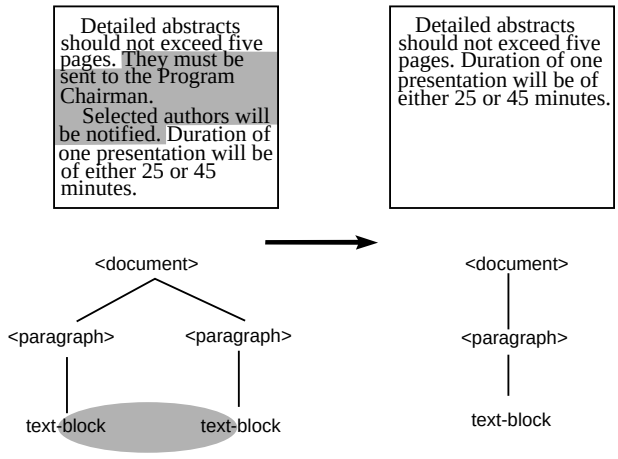

Figure 11. Deletion across like structures

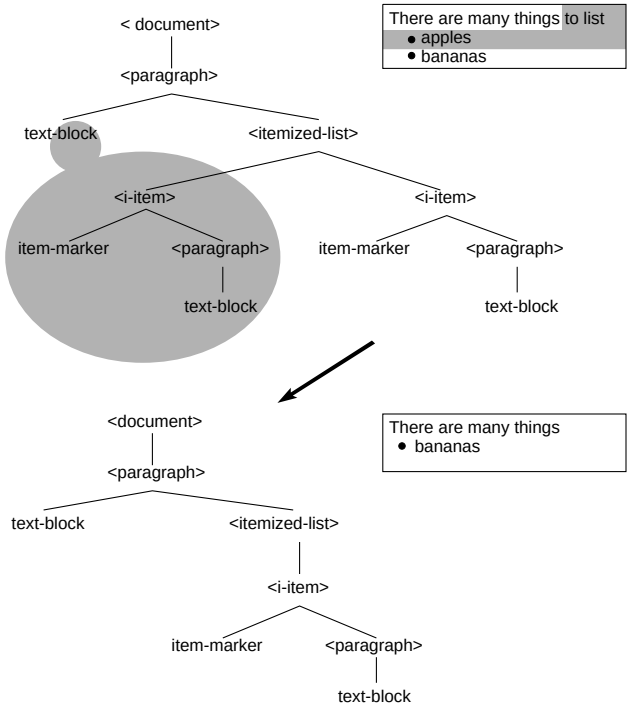

Figure 12. Deletion across unlike structures

heuristically-defined algorithms can be applied to emulate the effects of WYSIWYG operations.

Perhaps the most successful demonstration that such heuristically-defined operations can emulate the corresponding WYSIWYG operations is the **pedtnt**'s region deletion command. Document objects located within the region are classified as completely selected (the entire object is within the region), left-part selected (the region begins within the object), right-part selected (the region ends within the object), and center-part selected (the region both begins and ends within the object). Deletion when the object is center-part selected does not affect the structure—only the object itself. Completely selected objects are removed from the **tnt**, and hence the structure is affected. Left-part and right-part selected objects are not removed from the **tnt**, but may affect the structure since following deletion, the operation attempts to merge the left-part selected object with the right-part selected object. Merging is permitted only if the selected objects are of 'compatible' types (in effect, if the objects are of the same type).

Figures 11 and 12 illustrate the effects of the merging operation. The left hand side of Figure 11 shows a document fragment containing two paragraphs with a region beginning in the middle of the first paragraph and extending through the middle of the second. After deletion, the leaf text-blocks are combined and similarly the enclosing paragraphs are combined. The resulting document fragment is shown on the right. Figure 12, illustrates a case in which the objects are not combined. The upper illustration shows a region that begins within a text-block and extends into the following itemized-list, completely encompassing the first item of the itemized-list. Deletion preserves the distinction between the text block and the itemized-list.

CURRENT RESEARCH PROBLEMS AND LIMITATIONS

Active research continues into the development of these and and similar systems. Much of the work to date has concentrated on the logical representation of the document and the mapping between logical and physical representations. As these issues become better understood, attention is being paid to some important but secondary issues.

Given a grammatical specification of the generic structure of the document, an important unsolved problem is how to transform the form of an instance of the generic structure to correspond to a different specification. Such transformations arise either because the system's user wants to change the type of an object (and hence the object's contents are to be described by a different production in the grammar) or because corrections have been made to the grammatical description itself. Reparsing the component objects with the new target production will not necessarily succeed because the specifications are generally ambiguous. Simple redistribution schemes, such as those incorporated into some syntax-directed editors for programming languages, also will not necessarily provide correct results.

In the absence of an automatic solution, an approach that one of us (Furuta) is examining is to identify a formalism by which the relationships between the two document objects can be expressed[19]. The formalism will serve as the basis for development of a tool that will allow explicit specification of the relationships as well as providing a base upon which more automatic transformations may be built and tested.

A second issue of importance that remains to be addressed is the very practical one of how these systems will operate in the 'real world'. The systems that we have described

are oriented to use in the scientific community of which we are a part. It may be the case that further development will be necessary if such systems are to be used in significantly different environments. As one example, governmental document standards are frequently quite detailed and require extraordinarily complex and heavily nested structures. Supporting such documents may require adding levels of hierarchy to the user interface to keep it manageable.

As a practical matter, many documents exist already in other formats. While being able to import these other formats is clearly a desirable goal, it may be difficult to achieve, particularly if the document model to be imported is less structured than the models described here. The corresponding issue of export into other formats has been investigated within the scope of our projects. Grif, in particular incorporates a 'Translator' that can produce input for a number of batch-oriented formatter[11].

These systems incorporate a structured representation of the document, and consequently they may favor organized and structured specification of the document as well (for example, top-down specification). Identification and development of techniques that permit less-structured specification is important.

Finally, an important topic for future research will be to further explore the potential of the structured document representation and of the grammatical specification. Scribe permits its document description to be divided into multiple *parts*, each representing a contiguous segment of the document. Each part can be formatted individually, or all can be formatted at the same time. In either case, the output representation remains the same—cross references and pagination are maintained correctly.

The tree structure used as a document representation in the systems of this paper lends itself naturally to such subdivision of the document. Moreover, it permits development of tools that control concurrent access to the document and its parts by multiple authors—perhaps permitting multiple readers but only permitting a single writer to modify a given subtree.

It also will be fruitful to consider further use of secondary structures in the document representation. Perhaps such relationships could be used to associate data sources and filters with the document, perhaps permitting a document to show an algorithm, the input to the algorithm, and the output produced by the algorithm such that changes the algorithm or its input would automatically change the output. Indeed, the representation of the algorithm itself could be pretty-printed using such mechanisms.

Such considerations suggest that significantly more still can be done to increase the flexibility with which computer systems assist the author. Indeed, an important consideration in the development of these systems will not only be the tools that assist the author in creation of a document, but also the tools that assist the author in revision, maintenance, and reuse of the document and the tools that mediate between individuals in a group of authors when creating a common document.

HISTORY

Grif

Project Grif was launched in 1982, at the Laboratoire de Génie Informatique, University of Grenoble, with the support of INRIA. It was part of a larger project, called Tigre (Tiger in English), whose aim was to first define a model for structured documents, and then to

design and implement tools for handling documents: an interactive editor/formatter and a data base management system.

Between six and eight persons contributed to the Tigre model definition. This larger group was divided into two sub-groups, one for the editor (Grif) the other for the DBMS. Grif was designed by Irène Vatton and Vincent Quint. Vatton and Quint began implementing Grif in January 1984. Vatton was in charge of the specification and implementation of the 'Mediator', which is the user interface manager and the display manager. This version of the Mediator contains 11000 lines of Pascal and 4300 lines of C.

Quint specified and implemented the compilers and the editor itself. Grif incorporates two compilers—one for the grammatical definition of logical structures and the other for the presentation rules that specify the document's layout and the graphical aspects. These compilers produce tables that are used by the editor and total about 5000 lines of Pascal. The Grif editor is some 25 000 lines of Pascal.

Hassan Bedor participated in the Grif project from 1985 to 1986. Bedor specified and implemented the Translator used for translating Grif documents into the syntax of various formatters (**troff**, TEX, LATEX, Mint, and SGML). The Translator is 4500 lines of Pascal.

Until mid-1987, Grif development was carried out on a Perq running Unix, with its specific window manager. Grif has now been ported to a Sun-3 under the X window system, and a commercial firm in France has ported Grif to a PC-AT under MS-Windows and is using it as a basis for development of an ODA editor. The Grif-related figures in this paper were produced by the X window version.

pedtnt

The **tnt** document representation and the **pedtnt** editor were developed by Richard Furuta at the University of Washington beginning in the early 1980's. As a prototype intended as a testbed for the development of ideas, the **pedtnt**'s display is directed to a standard character-oriented computer terminal. A generalized manipulator, which modifies the strict tree portion of the **tnt**, is the center of the system. Also implemented are specialized editors for text-blocks, equation-blocks, and table-blocks. The implementation totals some 35 000 lines of C code, and runs under Berkeley Unix.

ACKNOWLEDGEMENTS

Richard Furuta is affiliated with the Department of Computer Science and the Institute for Advanced Computer Studies at the University of Maryland, College Park Campus. His work was supported in part by a grant from the General Research Board of the University of Maryland. Vincent Quint is affiliated with the Institut National de Recherche en Informatique et en Automatique and the University of Grenoble, Laboratoire de Génie Informatique. Jacques André is affiliated with the Institut National de Recherche en Informatique et en Automatique, Rennes laboratory (IRISA).

REFERENCES

1. *Document Composition Facility Generalized Markup Language: Concepts and design guide*. IBM Corporation, second edition, April 1980. Order number SH20-9188-0.

2. C. F. Goldfarb, 'A generalized approach to document markup', in *Proceedings of the ACM SIGPLAN SIGOA Symposium on Text Manipulation*, *SIGPLAN Notices*, 16(6):68–73, June 1981. The proceedings of the conference containing this paper are also available as *SIGOA Newsletter* 2(1&2), Spring/Summer 1981.
3. Brian K. Reid, *Scribe: A Document Specification Language and its Compiler*, PhD thesis, Carnegie-Mellon University Computer Science Department, Pittsburgh, PA, October 1980. Also issued as Technical Report CMU-CS-81-100.
4. Richard Furuta, Jeffrey Scofield, and Alan Shaw, 'Document formatting systems: Survey, concepts, and issues', *ACM Computing Surveys*, 14(3):417–472, September 1982.
5. *Text and Office Systems—Standard Generalized Markup Language*. ISO, October 1986. Document Number: ISO 8879–1986(E).
6. *Office Document Architecture*. International Standard Organisation, 1986. Draft International Standard 8813.
7. Jacques André, Richard Furuta, and Vincent Quint, eds. *Structured Documents*, Cambridge University Press, 1988. To appear.
8. Richard Furuta, 'An Integrated, but not Exact-Representation, Editor/Formatter', in J. C. van Vliet, ed. *Text Processing and Document Manipulation*, pages 246–259, Cambridge University Press, April 1986. Proceedings of the international conference, University of Nottingham, 14–16 April 1986.
9. Richard Furuta, *An Integrated, but not Exact-Representation, Editor/Formatter*, PhD thesis, University of Washington, Department of Computer Science, Seattle, WA, 1986. Also available as Technical Report No. 86-09-08, Department of Computer Science, University of Washington (August 1986).
10. Vincent Quint and Irène Vatton, 'GRIF: An interactive system for structured document manipulation', in J. C. van Vliet, ed. *Text Processing and Document Manipulation*, pages 200–213, Cambridge University Press, April 1986. Proceedings of the international conference, University of Nottingham, 14–16 April 1986.
11. Vincent Quint, Irène Vatton, and Hassan Bedor, 'Grif: An interactive environment for TEX', in Jacques Désarménien, ed. *TEX for Scientific Documentation*, pages 145–158, Springer-Verlag, 1986. Lecture notes in Computer Science, No. 236.
12. Vincent Quint, Irène Vatton, and Hassan Bedor, 'The Grif system', *TSI—Technology and Science of Informatics*, 6(1):98–103, April 1987.
13. Jonathan Seybold, 'Xerox's "Star"', *The Seybold Report*, 10(16), April 27, 1981.
14. David Canfield Smith, Charles Irby, Ralph Kimball, and Bill Verplank, 'Designing the Star user interface', *Byte*, 7(4):242–282, April 1982.
15. Robert A. Morris, 'Is what you see enough to get? A description of the Interleaf publishing system', in J. J. H. Miller, ed. *PROTEXT II: Proceedings of the Second International Conference on Text Processing Systems*, pages 56–81, Boole Press, October 1985.
16. *MacWrite*. Apple Computer, Inc., 1984.
17. Ben Shneiderman, 'Direct manipulation: A step beyond programming languages', *Computer*, 16(8):57–69, August 1983.
18. Vincent Quint and Irène Vatton, 'An abstract model for interactive pictures', in H.-J. Bullinger and B. Shackel, ed. *Human-Computer Interaction—INTERACT'87*, pages 643–647, North-Holland, 1987.
19. Richard Furuta and P. David Stotts, 'Specifying Structured Document Transformations', in J. C. van Vliet, ed. *Document Manipulation and Typography*, Cambridge University Press, April 1988, Proceedings of the Inernational Conference on Electronic Publishing, Document Manipulation and Typography, Nice (France), April 20–22, 1988.