ELECTRONIC PUBLISHING, VOL. 6(4), 327–337 (DECEMBER 1993)

Merging logical and physical structures in documents

CÉCILE ROISIN[1] AND IRÈNE VATTON[2]

INRIA-Imag
2, rue de Vignate
F-38610 Gières, France

email: Cecile.Roisin@imag.fr, Irene.Vatton@imag.fr

SUMMARY

Although it is well established that structured documents and generic models bring benefits to applications involving documents, integrating these document models in the formatting process of interactive editors is still an open problem. In this paper, the problem of laying out and formatting structured documents is investigated, taking into account the DSSSL standard. One key point of this model is the possibility of expressing the logical structure of documents independently from their graphical aspect. However, this approach induces a more complex formatting process, as two independent structures have to be merged. This discussion is illustrated by our experience of dynamic formatting in the Grif editor.

KEY WORDS Structured documents Interactive editing Formatting process

1 INTRODUCTION

It seems that a common agreement has been reached on document models such as those involved in the well-accepted standards SGML [1] or ODA [2] and the benefits of these models have been largely demonstrated [3]. Abstract document models are now used for many kinds of document-based applications: technical documentation, data bases, information retrieval, etc.

However, for the most popular and largely available application, namely document production, structured document models are seldom used with all their features. Most available products handle a linear document representation, in which content and layout information is mixed, thus preventing any application to take advantage of the complete logical representation of the edited document. This simple approach is sufficient for editing small, short life cycle and unstructured documents such as letters, memos, or short reports, but it is not suited to long life cycle and complex documents with heavy structural constraints such as technical documentation. For this kind of documents, structure manipulation and control are necessary.

An important reason that has limited the use of structured document models in document production is the difficulty of developing an editing tool with both logical and physical

[1] INRIA Rhône-Alpes
[2] CNRS-Imag

CCC 0894–3982/93/040327–11

Received 15 August 1993
Revised 1 December 1993

document representations. Mittelbach and Rowley have shown that the application context of typographical properties can come from two independent levels [4]: the structural level (the logical position of the element to be formatted) and the visual level (i.e., in the physical support). Some tools provide structured editing functionalities with poor formatting capabilities such as Author/Editor from SoftQuad, while others provide more sophisticated formatting operations but no interactive manipulation (e.g., LaTeX). The aim of this paper is to investigate the issues raised by mixing dynamic formatting and structured manipulation.

The paper is organized as follows. The next section analyses the needs for specifying and implementing document layout and presents the state of the art in this area. Section 3 enumerates the desired requirements for interactive edition and formatting. A method for implementing dynamic formatting in the Grif editor is then described in Section 4. The last section analyses, in the light of the experience gained with Grif, how standards can model dynamic formatting for structured documents.

2 PRINCIPLES OF FORMATTING

Among the typographical properties (or presentation properties) that characterize the graphical aspect of documents, we can identify two subsets:

1. Properties depending on the content to be laid out, like fonts, color, or typefaces. We call these properties the *style*.
2. Properties depending on the output medium, such as the size of pages, columns, margins, and gutters; we call these properties *physical structure* properties.

With a structured model of documents, the formatting process produces a representation of the document ready to be output (displayed or printed) from the internal representation of that document (its logical structure) and the style and physical structure properties. We analyse the typography expression and the formatting process in existing tools in the light of their ability to deal with both logical and physical levels.

The expression of document presentation has evolved in many directions, from low-level commands interspersed within the text (troff, TeX) to style sheets in interactive editors (Word, Author/Editor), or separate rules expressed in a specific language (Grif [5], Ensemble [6]). This evolution follows the evolution of document models, from unstructured (linear) or weakly structured document models[3] to structured document models, such as SGML Document Type Definitions (DTD).

One key point of structured document models is their ability to associate presentation properties with document element types, allowing inheritance of properties based on the structural hierarchy [7,8]. It is worth noting that only style properties can be related to the logical structure, unlike physical structure properties, which are independent from the logical structure.

Some systems manipulating weakly structured models propose some structuring features such as FrameMaker with its FrameBuilder tool, but these new structuring features do not change the editing and formatting processes. As a result, typographical properties cannot be defined as having logical contextual dependencies because they are associated with elements by way of style sheets: for instance, there is no way to express conditional rules.

[3] We consider models representing a document as a sequence of paragraphs with styles as weakly structured, as opposed to models involving tree structures and generic structures.

Editors based on fully structured models maintain a logical representation of the document which is used for editing operations and which can be more or less used by the formatting process, depending on the relationships between presentation properties and logical structure: from simple style sheets as in SoftQuad Author/Editor, where no complex dependencies can be defined, to style attributes inside the DTD [9,10] (suited for style properties, but not for physical structure properties), and finally to a separate expression of typographical properties with flowing rules associating these properties with the elements of the logical structure [7,8]. This last solution allows features not available with the style attributes, i.e., the ability to produce different graphical representations for the same document and to specify more complex physical structure models.

However, owing to the complexity in expressing and managing typographical contexts, these are still only partially taken into account in interactive structured documents formatters. Typographical properties can be very sophisticated and even contradictory to themselves [11]. Typical examples of complex formatting operations are floating objects placement, non-textual objects formatting (tables, equations, drawings), and global pages and lines balancing.

As a concluding remark of this quick overview, we can notice that when the logical structure needs only a linear placement of elements, which is typically the case in the list structure used in most textual documents, formatting properties can be expressed by style sheets and the formatting process is almost correctly performed by most existing tools. This can be explained by the fact that only one physical dimension has to be considered. The problem is much more complex when two-dimensional placements are needed, as in the case of tables or mathematical formulae. More complex techniques are then required.

3 REQUIREMENTS FOR INTERACTIVE FORMATTERS

Given the previous analysis, we can list the properties that must be provided to deal with both structured editing functionalities and powerful formatting capabilities. They are decomposed into two sub-sets: requirements for expressing typographical properties and requirements for the formatting process.

The expression of typographical properties must respect the two following characteristics:

- The logical and presentation models must be independent, allowing generic definitions for style and physical structure properties. The better way to deal with that point is to use a declarative language for expressing typographical properties.
- The presentation model must associate presentation properties with document element types, allowing inheritance of style properties based on the logical structure. The flowing model must allow one or many flows and one or many levels of formatting as in the models defined in ODA [2,12] and DSSSL [13].

The requirements for the formatting process come from the interpretation of powerful typographical properties on the one hand, and from constraints due to an interactive context on the other hand:

- The formatting process must deal with properties which depend on both the logical structure and the physical structure. The formatter must be able to solve possible conflicts between these properties, by means of a global function minimization as in

TEX [14], weighted properties as proposed in DSSSL [13], by an iterative process [15], or by cooperative formatters synchronized by a Block Manager as in Quill [9]. The method used must be compatible with interactive formatting performances.

- The formatting process must be generic enough to format any kind of objects (text, graphics, mathematical formulae) and consequently to allow mixed objects formatting: a formula can be formatted inside a text line, a paragraph can be put in line in a cell of a table. Methods based on content architectures such as proposed in ODA, Quill [4], or Proteus [7], cannot easily fit this requirement.
- It should be possible to format a document partially, this is especially true for large documents: during interactive editing, the formatting process should not be performed on the whole document, only the parts that have to be displayed should be formatted. This brings benefits in memory space and performance, but managing partially formatted documents introduces greater complexity into the editor/formatter.
- As formatting algorithms are time consuming, incremental formatting must be performed when the user modifies a document [8,12]. For example, when a paragraph is updated, only some lines of that paragraph need to be reformatted. This incremental formatting leads to an incremental display where only modified portions of the document have to be redisplayed, bringing better visual conveniences for the user than frequent redisplays of the full window.
- Editing and formatting operations must be directed by the logical model (implementing logical context dependencies) and the actions performed by the user must be correctly interpreted in dynamically updating the logical structure of the document.

There is not yet any available tool that fulfils all these requirements because of the extent of the problems to deal with. Implementing an interactive editor/formatter for structured documents is still an open challenge. However, theoretical work is going on [7,16,15], experiments are being carried out [6,12] and standards have been defined. The work presented in the next section describes our formatting experience based on the Grif editor: the presentation model and formatting process of this structured documents editor have been extended in order to manage more complex physical structures than in the previous prototype version. The objective of that experiment is clearly to obtain a tool that better fulfils the requirements mentioned above.

4 THE FORMATTING PROCESS IN GRIF

Grif is an interactive system for editing and formatting complex structured documents [8,5]. It must be seen as a basic research prototype onto which new research experiments are developed: index, spelling corrector [17], hypertext features [18], active document applications [19], static and dynamic document conversion [20], and, what is the core of this paper, managing complex physical structures in a dynamic formatting process.

As this paper focuses on formatting, the rest of the section is devoted to the formatting features of Grif in the light of the previous requirements, without describing other features of Grif.

The specific logical structure of a document is described by a generic logical structure (a DTD), with some extensions relatively to SGML such as the set of available basic types; in Grif, basic elements cannot only be of type text, but also of type graphic or picture.

```
<!ELEMENT Article -- (Front, Title, Author, Body) >
<!ELEMENT Body    -o (Section+)                   >
<!ELEMENT Section -o (Para+)                      >
<!ELEMENT Para    -o (Figure | #PCDATA | Table)   >
```

Figure 1. A DTD for an article

4.1 Presentation model in Grif

Grif allows the definition of generic typographical rules for any DTD. These rules are expressed separately from the DTD in a declarative language called P and are grouped in *presentation models*. A presentation model specifies the graphical appearance of all elements and attributes defined in a DTD. Figure 1 shows a simple example of a Document Type Definition for documents of type Article and Figure 2 is the partial presentation model for this DTD.

Presentation models contain rules which correspond to the two sub-sets of properties identified in Section 2:

- Style rules: they are associated to each element type of the DTD, allowing logical contextual dependencies. In the above example, the character size of element type Front is given by the rule: *Size : Enclosing . Size - 2 pt*; which means that the character size in a Front element is 2 pt smaller than in its parent in the tree.
- Layout rules: page models are defined (see the boxes C1, C2 and "*Simple_Page*" in the example) and the flowing rule *Page* states in which page model the elements have to be poured into.

Moreover, it is possible to associate *presentation boxes* to elements by means of creation rules such as *Create* (see the rule Create (Logo) for element type Article in the example). These boxes are automatically generated when the formatter displays the corresponding element; they may contain computed information (the section number for example), repeated content of an element (the title of the current section on top of each page), or a static value (the character string "abstract" added before the element abstract, a colored box, horizontal and/or vertical hairlines, etc.).

4.2 Overview of the editor/formatter

Since Grif is an interactive editor/formatter for structured documents, it has been designed to deal with both structured document representation and specific interactive constraints. Therefore, its main characteristics are [5]:

- Hierarchical data structures for textual and non textual document representation.
- Multiple views displaying: the same document can be displayed differently in many windows according to different needs (full document, table of contents, etc.).
- Partial formatting: only the parts of the document that must be displayed are formatted, so that the editor is equivalently adapted to any size of documents.
- Incremental formatting: all geometric dependencies between boxes are registered, allowing fast updating of the displayed picture.

```
PRESENTATION Article;
BOXES
  C1:                  { left column model }
      BEGIN
        Height: Enclosing . Height;
        Width: Enclosing . Width * 45%;
        HorizPos: Left = Enclosing . Left;
      END;
  C2:                  { right column model }
      BEGIN
        Height: Enclosing . Height;
        Width: Enclosing . Width * 45%;
        HorizPos: Left = Enclosing . Left
                         + Enclosing . Width * 55%;
      END;
  Simple_Page:         { page model }
      BEGIN
        Height: 25 cm;
        Width: 14 cm;
        Column (C1, C2);
      END;
  Logo:                { presentation box }
      BEGIN
        Content: Picture "INRIALogo";
      END;
RULES                  { presentation rules associated with
                         each element type of the DTD }
  Article:
      BEGIN
        Page(Simple_Page);     { flowing rule }
        Create (Logo);         { presentation box creation }
        Size: 12 pt;
        ...
      END;
  Front :
      BEGIN
        Size : Enclosing . Size - 2 pt;
      END;
  ...
END.
```

Figure 2. A (partial) presentation model for an article

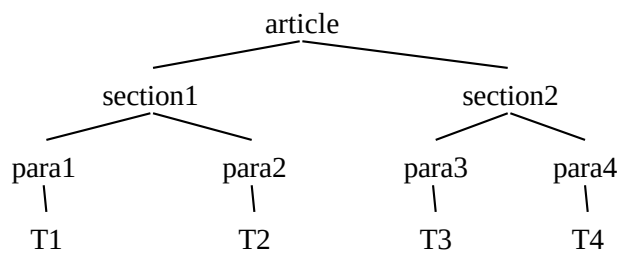

Figure 3. Abstract tree of an article

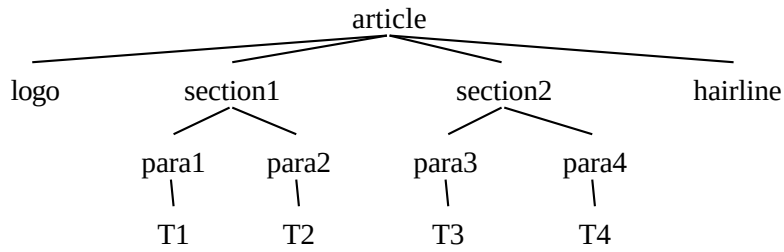

Figure 4. Abstract picture of an article before page computation

In Grif, formatting a document consists in transforming an *abstract tree* (the internal representation of a document) into its *concrete picture* (a set of boxes containing all information needed for displaying or printing the document). The formatting process is performed by two cooperating components called the *Editing Component* (EC), and the *Formatting Component* (FC), which share an intermediate representation of documents called an *abstract picture* (see Figure 4).

The abstract picture is a tree in which each node describes a rectangular box to be displayed with its associated typographical properties. There is one node in the abstract picture for each element in the abstract tree, if the element has to be displayed; this abstract picture is completed by *presentation boxes* defined in the presentation model. In the example of Figure 4, two presentation boxes are created for element Article: for instance, a logo in front of the article, and a hairline indicating the end of the article.

The EC constructs the abstract picture of the document by interpreting the presentation rules associated with the type of each node of the abstract tree (the rules defined in the presentation model). However, typographical properties are still uncomputed: they are expressed as constraints reflecting logical dependencies. The task of the EC is mainly to compute formatting operations which depend on the logical structure, i.e., what we have called style properties.

The FC computes the concrete picture, performing all the formatting operations depending on the physical structure which is defined by the layout rules in the presentation model: dividing the document into pages and columns, splitting text into lines with correct spacing and hyphenation. The resolution of placement constraints stored in the abstract picture is performed at this stage and the result is a set of boxes where each box delimits the area filled by the corresponding node.

However, this process is not just a one-way process from the EC to the FC. Some formatting operations depend both on the logical structure and on the physical structure. For instance, when a table is split by a page break, the knowledge of the logical structure is needed to produce a copy of the table heading on the new page. Floating objects in page headers or footers are other typical examples. Therefore, the abstract picture must take into account some information resulting from the evaluation of the FC; this is performed by a dialogue between the EC and the FC as explained in the next subsection for page formatting.

4.3 The formatting process for page construction

The flowing process, which pours the elements of the logical structure into the physical structure, is implemented as a two-step tree transformation applied to the abstract picture. In the following description, we call *logical nodes* the nodes issued from the logical structure (element nodes and presentation nodes) and *physical nodes* the nodes issued from the physical structure (pages, columns, etc.).

In order to illustrate page construction, we describe this data structure transformation with a document instance of type Article whose abstract tree is represented in Figure 3.

As explained in the previous subsection, the first formatting step is performed by the EC: the application of the style rules produces the abstract picture of Figure 4, with the same tree structure as the abstract tree plus additional presentation nodes (here logo and hairline) as stated in the presentation schema of Figure 2. This first version of the abstract picture can be considered as a galley that the FC, which knows the page size, has to split into pages during the second formatting step. In order to record this physical decomposition in the abstract picture, the FC tells the EC which nodes are on the page break in the abstract picture.

With this information, the EC creates the physical nodes (pages, columns) and hooks under them the logical nodes which are physically included (or poured) into them. This operation takes into account breaking rules defined in the presentation model (e.g., a new section cannot start at the bottom of the page). This operation of splitting the galley is performed step by step, where a step is a page (or a column for multi-columns pages). The formatting process is thus a synchronized cooperation between the FC and the EC.

In the example of Figure 2, the physical structure model used for formatting the document of Figure 3 specifies pages with two columns (see rule *Column(C1, C2)*), the result of the transformation of the abstract picture of Figure 4 is the tree of Figure 5:

- All physical nodes are set on top of the abstract picture: SetOfPages, PageHeader, PageFooter, PageBody, SetOfCols, and Cols.
- The logical nodes are placed as children of the physical nodes in which they will be displayed. When an element is spread on several physical areas, it is represented by several nodes in the abstract picture (for instance, article, article' and article" in Figure 5). As a result, nodes of the same element have the same style properties, but their position and dimension may be different. These nodes are chained together in order to maintain links between each element in the abstract tree and all its boxes.

Thanks to this resulting abstract picture, the formatting process can deal with both logical and physical dependencies.

As an example of more complex formatting, we can describe how floating objects are placed in page footers. The problem is to put the boxes corresponding to each floating element, such as figures or footnotes, in the footer of the page where the first reference to that element appears. Links between elements (here between one or many references to the referred element) are part of the abstract structure of the document managed by the EC [18]. Therefore, when the EC constructs the abstract picture, it can easily detect if the current element is the first reference to a floating element. If it is the case, it computes at that time the corresponding portion of the abstract picture and puts it under the current page footer node. In order to be sure that the page does not overflow with that floating element, the FC evaluates the new page boundary. If this boundary is above the floating element,

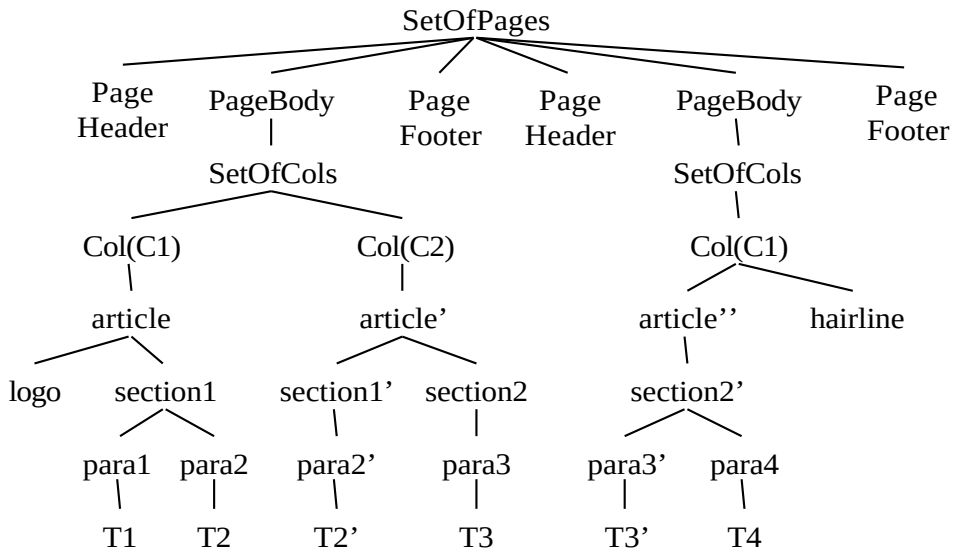

Figure 5. Abstract picture of a formatted article

the EC inserts a page break before the reference and therefore the abstract picture of the floating element will be moved to the next page. With this method, pages cannot be too long, but can be too short if floating objects are very long: more global strategies of element placement are needed to solve these cases (e.g., a footnote can be spread over two-page footers). However, we can notice that the key point of this kind of multiple flow pouring is a non-linear representation of the galley (i.e., the abstract picture).

5 DISCUSSION

The experience gained in implementing and using the Grif editor allows us to draw some conclusions about editing and formatting tools for structured documents:

- Some style properties need to be closely associated to the logical structure, specifically when structural contextual dependencies are required.
- In order to format elements whose placement depends both on the logical structure and the physical structure, a cooperation between the editing process and the formatting process is necessary.
- In an interactive editor/formatter, there is a crucial need not only for traversing the document elements from the logical structure to the physical areas, but also for the reverse direction, i.e., given a physical area on the screen, to find to which structure elements it corresponds. In Grif, the abstract picture is the data structure that maintains these bidirectional links (between the abstract tree and the concrete picture).

If we analyse our implementation in the light of standards, it is clear that our approach is closer to DSSSL than to ODA, mainly because the content architecture defined in ODA [12] prevents integrated formatting.

The architecture defined in DSSSL proposes a two-step process: the transformation process and the formatting process. In our implementation, we have also decomposed formatting in two steps. This clearly comes from the necessity to deal, in the first step, with presentation operations that depend on the logical structure, and, in the second step, with operations that depend on the layout structure. More precisely, this decomposition involves two processes:

1. The transformation process: it transforms the document, i.e., the source instance in DSSSL terminology, by applying such style rules as: adding presentation boxes, computing all information relevant to the structure, duplicating and/or suppressing information. The result is a tree called the output instance in DSSSL and called the abstract picture in Grif.
2. The formatting process: it produces the formatted document, the result instance in DSSSL, by instantiating physical areas, area templates in DSSSL, and pouring the output instance into these areas with respect to the flowing model. In our implementation, this operation is performed by hooking under each instantiated area the subtree of the output instance that fills it.

With this analysis, our implementation can be viewed as a first experience that conforms to the DSSSL architecture. Our main question about the adequacy of the DSSSL architecture to interactive structured edition/formatting is the lack of cooperation between the two processes. Another question is the ability to express structurally dependent properties, because presentation properties appear in the area definition, even if they do not rely on the physical structure. The DSSSL_ATTR specified in the output model could be a way to specify such properties.

REFERENCES

1. *International Standard ISO 8879: Information Processing – Text and Office Systems – Standard Generalized Markup Language (SGML)*, ed., International Organization of Standardization, Geneva/New York, First edition, 1986.
2. *International Standard ISO 8613: Information Processing – Text and Office Systems – Office Document Architecture (ODA) and Interchange Format*, ed., International Organization of Standardization, Geneva/New York, 1986.
3. J. André, R. Furuta, and V. Quint, *Structured documents*, Cambridge University Press, Cambridge, UK, 1989.
4. F. Mittelbach and C.A. Rowley, 'The pursuit of quality', in *EP92 Text processing and document manipulation*, eds., C. Vanoirbeek and G. Corey, pp. 77–94, Cambridge, UK, (April 1992). Cambridge University Press.
5. V. Quint and I. Vatton, 'An abstract model for interactive pictures', in *Human-Computer Interaction – Interact'87*, eds., H.-J. Bullinger and B. Shackel. Elsevier Science Publishers (North-Holland), (1987).
6. S.L. Graham, M.A. Harrison, and E.V. Munson, 'The Proteus Presentation System', in *Fourth ACM SIGSOFT Symposium on Software Development Environments*. Tyson's Corner, VA, (December 1992).
7. A.L. Brown Jr., T. Wakayama, and H.A. Blair, 'A reconstruction of context-dependent document processing in SGML', in *EP92 Text processing and document manipulation*, eds., C. Vanoirbeek and G. Corey, pp. 1–25, Cambridge, UK, (April 1992). Cambridge University Press.
8. R. Furuta, V. Quint, and J. André, 'Interactively editing structured documents', *Electronic Publishing*, **1**(1), 19–44, (April 1988).

9. D. Chamberlin, 'Managing properties in a system of cooperating editors', in *EP90 Text processing and document manipulation*, ed., R. Furuta, pp. 31–46, Cambridge, UK, (September 1990). Cambridge University Press.
10. D.D. Cowan, E.W. Mackie, G.M. Pianosi, and G. de V. Smit, 'Rita – an editor and user interface for manipulating structured documents', *Electronic Publishing – Origination, Dissemination, and Design*, **4**(3), 125–150, (September 1991).
11. R. Southall, 'Presentation rules and rules of composition in the formatting of complex text', in *EP92 Text processing and document manipulation*, eds., C. Vanoirbeek and G. Corey, pp. 275–290, Cambridge, UK, (April 1992). Cambridge University Press.
12. M. Murata and K. Hayashi, 'Formatter hierarchy for structured documents', in *EP92 Text processing and document manipulation*, eds., C. Vanoirbeek and G. Corey, pp. 77–94, Cambridge, UK, (April 1992). Cambridge University Press.
13. *International Standard ISO/DIS 10179: Information Technology – Text and Office Systems – Document Style Semantics and Specification Language (DSSSL)*, 1991.
14. D. Knuth, *The TEX Book*, Addison-Wesley, Reading, 1984.
15. F. Mittelbach and C.A. Rowley, 'A general model of document formatting', in *1st Workshop on Principles of Document Processing*, Washington, DC, (October 1992).
16. B.S. Hansen, 'A function-based formatting model', *Electronic Publishing – Origination, Dissemination, and Design*, **3**(1), 3–28, (February 1990).
17. H. Richy, P. Frison, and E. Picheral, 'Multilingual string-to-string correction in Grif, a structured editor', in *EP92 Text processing and document manipulation*, eds., C. Vanoirbeek and G. Corey, pp. 183–198, Cambridge, UK, (April 1992). Cambridge University Press.
18. V. Quint and I. Vatton, 'Combining hypertext and structured documents in Grif', in *ECHT'92*, ed., D. Lucarella, pp. 23–32, Milan, (December 1992). ACM Press.
19. V. Quint and I. Vatton, 'Making structured documents active', *Electronic Publishing – Origination, Dissemination, and Design*, (1994). to appear.
20. E. Akpotsui and V. Quint, 'Type transformation in structured editing systems', in *EP92 Text processing and document manipulation*, eds., C. Vanoirbeek and G. Corey, pp. 27–41, Cambridge, UK, (April 1992). Cambridge University Press.