INRIA

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

Formatting Structured Documents

Cécile ROISIN
Irène VATTON

N° 2044
Septembre 1993

PROGRAMME 3

Intelligence artificielle,
systèmes cognitifs et
interaction homme-machine





Formatting Structured Documents

Cécile Roisin*, Irène Vatton*

Programme 3 — Intelligence artificielle, systèmes cognitifs
et interaction homme-machine
Projet Opéra

Rapport de recherche n ° 0000 — Septembre 1993 — 21 pages

Abstract:

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 to express 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.

(Résumé : tsvp)

*{Cecile.Roisin}{Irene.Vatton}@imag.fr

Unité de recherche INRIA Rhône-Alpes
46 avenue Félix Viallet, 38031 GRENOBLE Cedex 1 (France)
Téléphone : (33) 76 57 47 77 – Télécopie : (33) 76 57 47 54

Le formatage des documents structurés

Résumé :

S'il est clairement établi que l'utilisation de modèles de documents structurés constitue une avancée significative dans le domaine des applications de manipulation de documents, l'intégration de ces structures de documents dans les formateurs interactifs est encore mal résolue. Ce rapport traite de ce problème en s'appuyant sur la norme DSSSL. Un des atouts de la structuration est la possibilité d'exprimer la structure logique des documents indépendamment de leur aspect graphique. Cette approche conduit cependant à la mise en œuvre d'un processus de formatage plus complexe, dans la mesure où il est nécessaire de fusionner deux structures indépendantes. Cette discussion est illustrée par notre expérience de formatage dynamique que nous avons réalisé dans l'éditeur Grif.

Mots-clé : documents structurés, édition interactive, processus de formatage.

1 Introduction

It seems that a common agreement has been reached on document models such as those involved in the well-accepted standards SGML [11] or ODA [10] and the benefits of these models have been largely demonstrated [2]. Current research in this area is mainly devoted to formal modelling (for example to perform automatic transformations upon documents [1]) or extensions in order to take into account hypertext and multimedia features [12].

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 keep the complete logical representation. 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 absolutely necessary.

An important reason that has limited the use of structured document models in document production is the difficulty to develop an editing tool with both logical and physical document representations. 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 two next sections analyse the needs for specifying and implementing document layout and present the state of the art in this area. Section 4 enumerates desired requirements for interactive edition and formatting. A method for implementing dynamic formatting in the Grif editor is then described in section 5. The last section analyses, in the light of the experience gained with Grif, how standards can model dynamic formatting for structured documents.

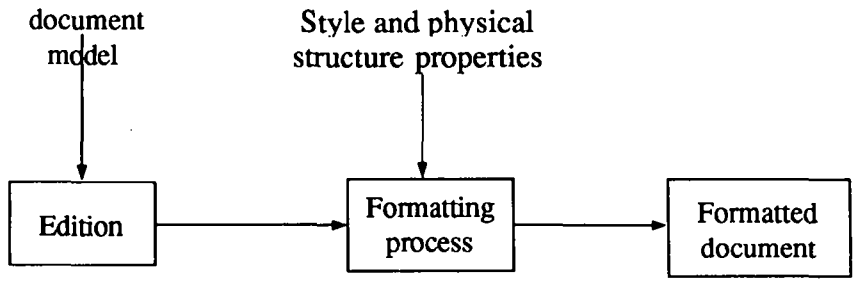


Figure 1: Formatting process

2 Principles of formatting

Formatting process produces a representation of the document ready to be output (displayed or printed) from the internal representation of that document and the style and physical structure properties (Fig. 1).

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

1. Properties depending on the content to be layed 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 them *physical structure* properties.

MittelBach and Rowley have shown that the application context of typographical properties can come from two independent levels [17]: the structural level (the logical position of the element to be formatted), and the visual (or physical) level (i.e. in the physical support). As the quality of formatting strongly depends on the ability to deal with these levels of dependencies, we analyse formatting principles and existing tools in the light of this criterion.

However, thanks to the complexity in expressing and managing typographical contexts, they are still only partially taken into account in interactive structured documents formatters. Typographical properties can be very sophisticated and even contradictory between themselves [23]. Typical examples of complex formatting operations, are floating objects placement,

non-textual objects formatting (tables, equations, drawings) and global pages and lines balancing. Therefore, algorithms for automatic placement are very complex and not yet full satisfactory; they are still under investigation (see [3], [18]).

3 Overview of structured document formatters

In this part, only interactive formatters for structured documents are considered. They are analysed from the point of view of the relationships between logical structures and physical structures.

The expression of document presentation has evolved in many directions, from low-level commands interspersed with the text (troff, TEX) to style sheets in interactive editors (Word, Author/Editor), or separate rules expressed in a specific language (Grif [20], Ensemble [8]). This evolution follows the evolution of document models, from unstructured (linear) or weakly structured document models[1] 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 [3], [7]. It is worth noting that only style properties can be related with the logical structure, unlike physical structure properties, which are independent from the logical structure.

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

Editors based on fully structured model maintain a logical representation of the document which is used for editing operations. For example inserting

[1] 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.

an element is allowed only if the result complies with the document model. In these editors, the formatting process can deal with the logical representation in one of the following ways, depending on the relationships between presentation properties and logical structure:

- Associating style sheets to each element type of the DTD, as in SoftQuad Author/Editor: the formatting process is close to those of unstructured editors, with similar drawbacks (no complex dependencies can be defined).

- Adding typographical properties in the logical structure by means of attributes: the logical hierarchy can be turned to account for setting style property dependency or inheritance: an element can inherit style properties from its ascendant elements [5], [6]. This mechanism is suited for style properties but not for physical structure properties such as page layout, as they do not rely on logical elements. Moreover, as logical attributes are integrated within documents, it is not possible to modify any typographical aspect of a document without changing the its content: logical attributes do not guarantee independency between logical structure and physical structure.

- Expressing typographical properties out of the DTD: this separation allows formatters to produce different graphical representations for the same document. For example, the same information printed at eleven point size or printed at twelve point size will be considered as a unique document. The association between properties and element types is either performed implicitly (by name homonymy such as in Grif [7]) or by explicit flow properties (as specified in [3] and in ODA and DSSSL). In addition, a physical structure model (unlinked with the DTD) can be specified, allowing complex layout definitions. The main difficulty of this approach is to implement a formatter which deals with both structures.

As a concluding remark of this quick overview, we can notice that when the logical structure needs only a linear placement of elements (that is typically the case of the list structure used in most textual documents), formatting properties by style sheets and formatting process are almost correctly performed by most existing tools. This can be explained by the fact than

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 technics are then required.

4 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 formatting process.

4.1 Requirements for expressing typographical properties

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 that point is to use a declarative language for expressing typographical properties. Like Grif [20], Ensemble [8] has defined a specific language that defines graphical results to be reached, instead of specifying how to reach them. In [3], Brown and al. have defined extensions of SGML that could allow the definition of graphical properties out of the DTD.

- The presentation model must associate presentation properties with document element types, allowing inheritance of style properties based on the logical structure [7]. The flowing model must allow one or many flows and one or many levels of formatting like in the models defined in ODA [10], [19] and DSSSL [13].

4.2 Requirements for formatting process

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

- The formatting process must deal with properties depending from 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 [15], weighted properties as proposed in DSSSL [13], iterative process [18], or cooperative formatters synchronized by a Block Manager as in Quill [16]. The method used must be compatible with interactive formatting performances.

- The formatting process must be generic enough in order 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 [5] or Proteus [8], cannot easily fit this requirement.

- It should be possible to format a document partially, especially for large documents: during interactive editing, the formatting process does not need to be performed on the whole document, only the parts that have to be displayed need to be formatted. That brings benefits in memory space and performance; but managing partial formatted documents induces more complexity in the editor/formatter.

- As formatting algorithms are time consuming, incremental formatting must be performed when the user modifies a document [7], [9], [19]. For example, when a paragraph is updated, only some lines of that paragraph need to be reformatted. This incremental formatting leads to an incremental displaying where only modified portion 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.

```
{ DTD Article }
<!ELEMENT Article -- (Front, Title, Author, Body) >
<!ELEMENT Front  -- (Date & Number & Version) >
<!ELEMENT Body -o (Para+) >
<!ELEMENT Para -o (Figure | #PCDATA | Table) >
```

Figure 2: A DTD for Article

4.3 Current status

Presently, there is not yet any available tool that fulfils all these requirements because of the extent of the problems to deal with. Implementing a structured documents interactive editor/formatter is still an open challenge. However, theoretical work is going on [3], [9], [18], experiences are carried out [8], [19] 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 the 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 fulfils better the requirements mentioned above.

5 The formatting process in Grif

Grif is an interactive system for editing and formatting complex structured documents [7], [20].

The specific logical structure of a document is described by a generic logical structure, a SGML DTD with some extensions such as the set of available basic types; in Grif, basic elements can be not only of type text but also of type graphic or picture. Fig. 2 shows a simple example of a Document Type Description for documents of type Article:

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 this tool (see [1], [21] and [22] for more details).

5.1 Presentation model in Grif

Grif allows to define generic typographical rules for any DTD. These rules are expressed separately from the DTD in a declarative language called P (P for Presentation) and are grouped in *presentation models.* A presentation model specifies the graphical aspect of all elements and attributes defined in a DTD. Fig. 3 is a partial presentation model for the DTD of Fig. 2.

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

- Style rules 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. The *visibility* rule defines filters on the document, allowing to display or not display some elements.

- Layout rules: page models are defined (see the box "*Simple_Page*" of the example) and the flowing rule *Page* expresses in which page model the elements have to be poured.

Moreover, it is possible to associate to elements *presentation boxes* 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 display 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 in the background, horizontal and/or vertical hairlines, etc.).

5.2 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 [20]:

- hierarchical data structures for textual and non textual document representation.

```
PRESENTATION Article;
{ DTD for which thispresentation model is defined }

BOXES                { pages models andpresentation boxes }
  C1: BEGIN          { left column model }
      Height: Enclosing . Height;
      Width: Enclosing . Width * 45%;
      HorizPos: Left = Enclosing . Left;
      END;
  C2: BEGIN          { right column model }
      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;
      VertPos: Top = Enclosing . Top + 2 cm;
      HorizPos: Left = Enclosing . Left + 3 cm;
      Column (C1, C2);
      END;
  Logo : BEGIN
      Content : Picture "INRIALogo";
      END;
RULES
  Article : BEGIN
      Page(Simple_Page);    { flowing rule }
      Create (Logo);        { presentation box }
      Size: 12 pt;
      END;
  Front : BEGIN
      Size : Enclosing . Size - 2 pt;
      Adjust: Right;
      END;
```

Figure 3: A (partial) presentation model for an Article

- Multiple views displaying: the same document can be displayed differently in many windows according to different needs (full document, table of content, 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 registering, allowing local updating of the display.

In Grif, formatting a document consists in transforming an *abstract tree* (the internal representation of a SGML document) into its *concrete picture* (a set of boxes containing all information needed for displaying the document). The formatting process is performed by two cooperating components called the *editing component* and the *formatting component* which share an intermediate representation of documents called an *abstract picture* (seeFig. 4).

The abstract picture is a tree in which each node describes a rectangular box to be displayed with its associated typographical properties. The tree structure reflects the area inclusion hierarchy.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 nodes* (the "pres" nodes in Fig. 4) defined in the presentation model. In the example of Fig. 6, 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 editing component 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 editing component is mainly to compute formatting operations which depend on the logical structure, i. e. what we have called style properties.

The formatting component 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

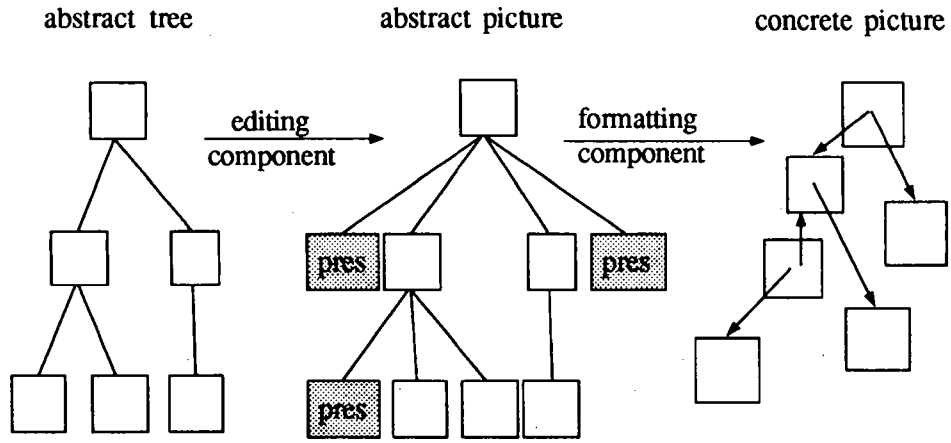


Figure 4: Data structures in the Grif editor

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 editing component to the formatting component. Some formatting operations depend both on logical structure and on physical structure. For instance, when tables are split by a page break, the knowledge of 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 formatting component; this is performed by a dialogue between the editing component and the formatting component as explained in the next subsection 5.3 for page formatting.

During interactive editing, the abstract picture is only computed partially (partial formatting) and is modified dynamically with respect to user operations (insertion, deletion, scrolling, browsing through the document with links, etc.).

Incremental formating is obtained by registering in the concrete picture dependencies and reverse dependencies of position and dimension between boundaries and axis of boxes. Therefore, when a box is moved or changed by a user action, the formatting component can locally compute again the modified properties of the neighbour boxes. Moreover, when an element

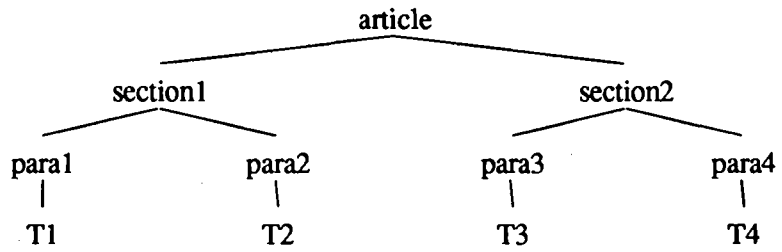


Figure 5: Abstract tree of an article

is modified, destroyed, or when a new element is inserted in the abstract tree, the editing component computes again the constraints for position and dimension of neighbour nodes and notifies to the formatting component what have changed.

5.3 The formatting process for pages construction

The flowing process, which pours the elements of the logical structure into the physical structure, is implemented as a 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 pages construction, we describe this data structure transformation with a document of type Article whose abstract tree is represented in Fig. 5.

As explained in the previous subsection, the first formatting step is performed by the editing component: the application of the style rules products the abstract picture of Fig. 6, with the same tree structure as the abstract tree plus additional presentation nodes (but nodes represent area with their typographical rules).

This first version of the abstract picture can be considered as a galley that the formatting component, 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 formatting component tells the editing component which nodes are on the page break in the abstract picture.

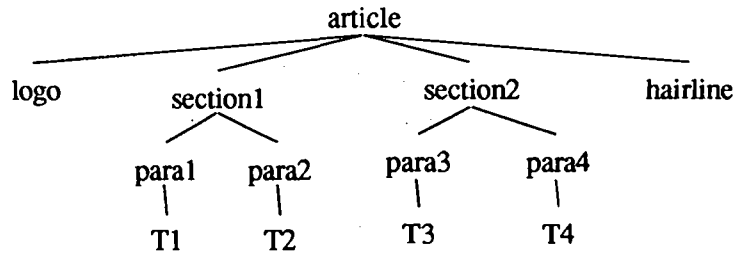


Figure 6: Abstract picture of an article before page computation

With this information, the editing component 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 formatting component and the editing component.

If we use the physical structure model of Fig. 3, which specifies pages with two columns (see rule *Column(C1, C2)*), for formatting the document of Fig. 5 , the result of the transformation of the abstract picture of Fig. 6 is the tree of Fig. 7:

- 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 area, it is represented by several nodes in the abstract picture (for instance, article, article' and article" in Fig. 7). 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.

This cooperation between the editing component and the formatting component products a shared structure (the abstract picture) in which informa-

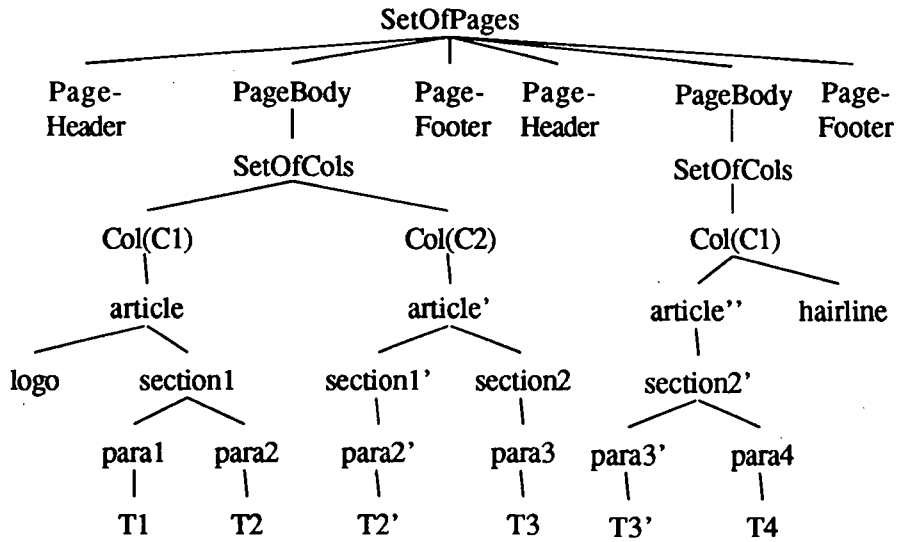


Figure 7: Abstract picture of a formatted article

tion from both logical and physical structures is stored. Thanks to the 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 editing component [21]. Therefore, when the editing component 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 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 formatting component evaluates the new page boundary. If this boundary is above the floating element, the editing component 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 through 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).

6 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. The two main differences between our approach and ODA model are the following:

- The content architecture defined in ODA prevents an integrated formatting of mixed objects. Objects of different nature are supposed to be formatted by specific tools [19]. However, it is clear that the formatter must be intensively generic to achieve an integrated formatting of objects of different nature. In the case of Grif, the presentation rules and the data structures managed by the editor are generic enough to deal with text as well as tables, mathematical formulae or graphics.

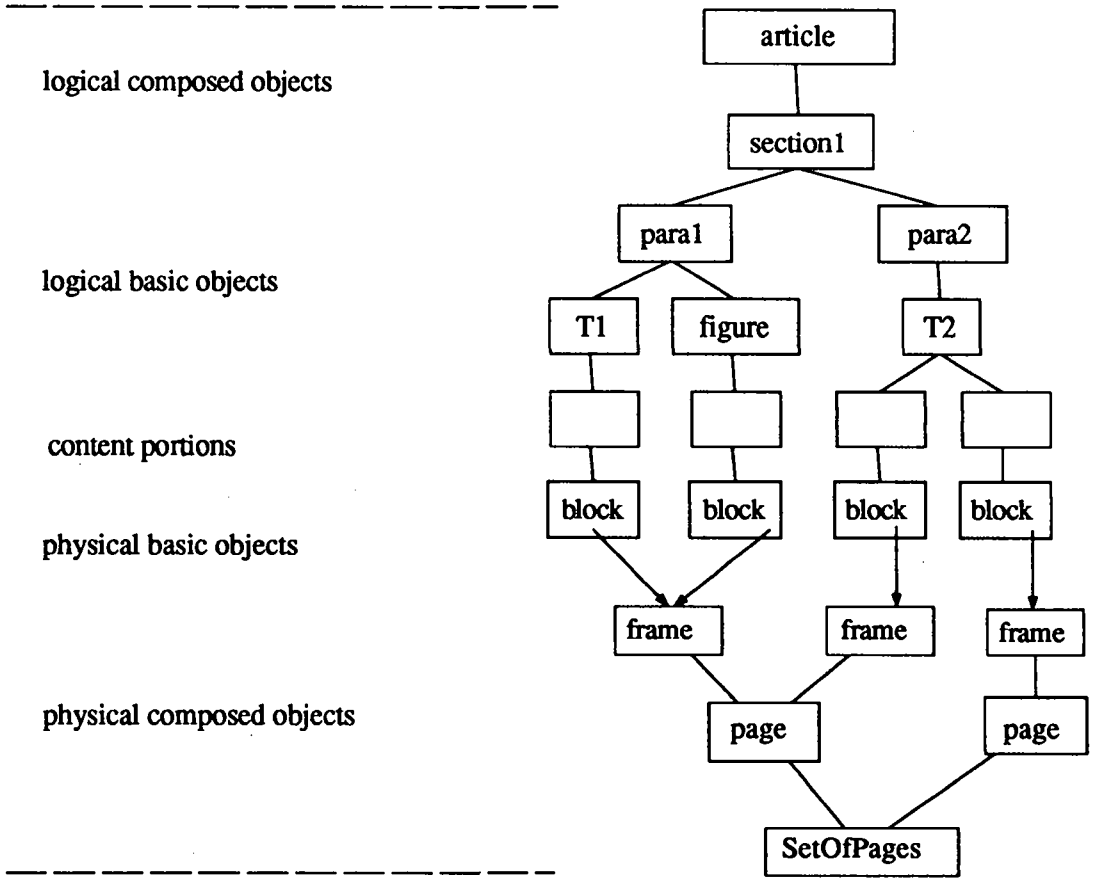


Figure 8: ODA flowing model

- The flowing process is performed by sharing the common leaves of the physical structure and the logical structure, as represented in Fig. 8. It is not clear how interactive edition can be performed with such a structure.

The architecture defined in DSSSL proposes a two steps process: the transformation process and the formatting process. In our implementation, we have also decomposed the formatting in two steps (cf. section 5.3). 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, called the source instance in the DSSSL vocabulary, 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.

DSSSL does not specify how these processes have to be performed. With this analysis, our implementation can be viewed as a first experience that conforms to DSSSL architecture.

However, some remarks can be formulated from this analysis:

- In order to format elements to which the placement depends both on the logical structure and the physical structure, a tightly cooperation between the transformation process and the formatting process is necessary. It is not clear how the mechanism of weighted placement rules as provided in DSSSL will be a solution for that specific problem.

- In an interactive editor/formatter, there is a crucial need not only for traversing the document elements from the logical structure to the result instance, but also for the reverse sense, i. e. given a physical area on the screen, to which structure elements does it correspond ? In Grif, the abstract picture is the data structure which maintains these bidirectional links (from the abstract tree to the concrete picture).

- Some style properties need to be closely associated to the logical structure, specifically when structural contextual dependencies are required. In DSSSL, 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] E. Akpotsui, V. Quint, "Type Transformation in Structured Editing Systems", *EP'92 Text processing and document manipulation,* C. Vanoirbeek & G. Corey, ed., pp. 27-41, Cambridge University Press, Cambridge, April 1992.

[2] J. André, R. Furuta , V. Quint, *Structured documents,* Cambridge University Press, Cambridge, 1989.

[3] A. L. Brown, Jr. Toshiro Wakayama, H. A. Blair, "A Reconstruction of Context-Dependent Document Processing in SGML", *EP'92 Text processing and document manipulation,* C. Vanoirbeek & G. Corey, ed., pp. 1-25, Cambridge University Press, Cambridge, April 1992.

[4] H. Brown, "Parallel processing and document layout", *Electronic Publishing,* vol. 1, num. 2, pp. 97-104, September 1988.

[5] D. Chamberlin, "Managing Properties in a System of Cooperating Editors", *EP'90 Text processing and document manipulation,* R.Furuta, ed., pp. 31-46, Cambridge University Press, Cambridge, September 1990.

[6] D. D. Cowan, E. W. Mackie, G. M. Pianosi, G. de V. Smit, "Rita-an editor and user interface for manipulating structured documents", *Electronic Publishing,* vol. 4, num. 3, pp. 125-125, September 1991.

[7] R. Furuta, V. Quint, J. André, "Interactively Editing Structured Documents", *Electronic Publishing,* vol. 1, num. 1, pp. 19-44, April 1988.

[8] S. L. Graham, M. A. Harrison, E. V. Munson, "The Proteus Presentation System", *Fourth ACM SIGSOFT Symposium on Software Development Environments, Tyson's Corner, VA,* December 1992.

[9] B. S. Hansen, "A Function-based Formatting Model", *Electronic Publishing,* vol. 3, num. 1, pp. 3-28, 1990.

[10] International Standard ISO 8613, *Information Processing - Text and Office Systems - Office Document Architecture (ODA) and interchange format,* International Standard Organization, 1986.

[11] International Standard ISO 8879, *Information Processing - Text and Office Systems - Standard Generalized Markup Language (SGML),* International Standard Organization, 1986.

[12] International Standard ISO Draft 10744, *Information Technology - Text and Hypermedia/Time-based Structuring Language (HyTime),* International Standard Organization, 1991.

[13] International Standard ISO Draft 10179, *Document Style Semantics and Specification Language (DSSSL),* International Standard Organization, 1991.

[14] V. Joloboff, "Trends and Standards in Document Representation", *Text processing and document manipulation,* J. C. van Vliet, ed., pp. 107-124, Cambridge University Press, Cambridge, April 1986.

[15] D. Knuth, *The TEX Book,* Addison-Wesley, Reading, 1984.

[16] A. W. Luniewski, "Intent-Base Page Modelling Using Blocks in the Quill Document Editor", *EP'88 Text processing and document manipulation,* J. C. van Vliet, ed., pp. 205-221, Cambridge University Press, Cambridge, April 1988.

[17] F. Mittelbach, C. A. Rowley, "The Pursuit of Quality", *EP'92 Text processing and document manipulation,* C. Vanoirbeek & G. Corey, ed., pp. 77-94, Cambridge University Press, Cambridge, April 1992.

[18] F. Mittelbach, C. A. Rowley, "A General Model of Document Formatting", *1 st Workshop on Principles of Document Processing,* Washington, October 1992.

[19] M. Murata, K. Hayashi, "Formatter Hierarchy for Structured Documents", *EP'92 Text processing and document manipulation,* C. Vanoirbeek & G. Corey, ed., pp. 77-94, Cambridge University Press, Cambridge, April 1992.

[20] V. Quint, I. Vatton, "An Abstract Model for Interactive Pictures", *Human-Computer Interaction - Interact'87,* H.-J. Bullinger & B. Shackel, ed., Elsevier Science Publishers B. V. (North-Holland), 1987.

[21] V. Quint, I. Vatton, "Combining Hypertext and Structured Documents in Grif", *Proceedings of ECHT'92,* D. Lucarella, ed., pp. 23-32, ACM Press, Milan, December 1992.

[22] H. Richy, P. Frison, E. Picheral, "Multilingual String-to-String Correction in Grif, a Structured Editor", *EP'92 Text processing and document manipulation,* C. Vanoirbeek & G. Corey, ed., pp. 183-198, Cambridge University Press, Cambridge, April 1992.

[23] R. Southall, "Presentation Rules and Rules of Composition in the Formatting of Complex Text", *EP'92,* C. Vanoirbeek & G. Coray, ed., pp. 275-290, Cambridge University Press, Cambridge, April 1992.



Unité de Recherche INRIA Rhône-Alpes
46, avenue Félix Viallet - 38031 GRENOBLE Cedex (France)

Unité de Recherche INRIA Lorraine Technopôle de Nancy-Brabois - Campus Scientifique
615, rue du Jardin Botanique - B.P. 101 - 54602 VILLERS LES NANCY Cedex (France)
Unité de Recherche INRIA Rennes IRISA, Campus Universitaire de Beaulieu 35042 RENNES Cedex (France)
Unité de Recherche INRIA Rocquencourt Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)
Unité de Recherche INRIA Sophia Antipolis 2004, route des Lucioles - B.P. 93 - 06902 SOPHIA ANTIPOLIS Cedex (France)

EDITEUR
INRIA - Domaine de Voluceau - Rocquencourt - B.P. 105 - 78153 LE CHESNAY Cedex (France)

ISSN 0249 - 6399

