Document Style Design by Direct Manipulation

Hélène Richy

Irisa, Campus universitaire de Beaulieu, F-35042 Rennes Cedex, France

Abstract. Designing style sheets for structured documents is often a difficult task. In this paper, we discuss the way to support style design through direct manipulation and propose an interactive method of specification by example for editing style sheets. In this approach, style editing actions within a formatted document are generalized into "generic" style specifications and the "generic" style sheet is dynamically updated.
An initial implementation of a direct-manipulation editor for structured document style sheet is presented. Based on a structured authoring environment, this prototype provides a comfortable environment for editing style properties as defined in style sheets through the visual representation of any document, without programming the style language.

1 Introduction

The separation of concerns between authoring, editing, designing, and typesetting was well established in the traditional publishing industry. In electronic publishing, a similar separation leads us to consider logically structured documents separated from their style specification. In this context, the document style design raises two questions. The first one relates to **design**, i.e. what will the document look like? No system will automatically create a perfect document design, since it is a matter of taste, aesthetics, or artistic creation. However, a comfortable environment using direct manipulation can assist this design process. The second question relates to **style specification**: how does the abstract design, or style sheet, ensure an accurate design?

Let us focus on electronic documents. Languages, such as DSSSL [9], or CSS [14], or soon XSL [1] are now available for style specification, and achieve better-quality layout, while allowing more sophisticated formatting operations. However, no support is available to define these specifications. Interfaces based on forms as provided by most word processing systems (*Microsoft Word*, *FrameMaker* [5], ...) for updating style properties are not sufficient to define sophisticated typographical properties [20] when these can be defined with logical contextual dependencies. Even with a declarative language like P [17] or CSS, writing style specifications for documents of a complex structure is an arduous task.

This situation calls for new features to be provided by editing environments for editing style sheets. Specifying by example seems a natural way to approach these problems [4] [16]. Thus, we propose a graphical style editing environment as an extension of a structured authoring environment [6] [7]. The goal of this system is to provide an environment for interactive style design of structured documents which may also produce the abstract style specification ensuring that similar documents will look the same.

We consider **direct manipulation** in style editing as being based on the following principles:

- abstract style specification is controlled by **visual checking on the formatted document** (interactive formatting),
- abstract style specification is inferred from **local style editing** within the formatted document,
- conditional style specification is indicated by **direct selection** of context within the formatted document.

This paper focuses on a style editing environment for structured documents. Section 2 first presents an overview of models and of design approaches. The third and fourth sections then present the proposed approach and discuss the merits and limitations of the presentation graph which forms its basis. Finally, some aspects of the initial prototype implemented on the Thot editor-formatter [2] are described in section 5, before the conclusions are presented in section 6.

2 Style Sheets and Structured Documents

2.1 Separation Between Structure and Presentation

Most desktop publishing systems use style sheets. Style sheets describe how documents will be formatted and transformed for printing or for screen display. Style sheets offer users a powerful way to provide an attractive presentation of their electronic documents.

Just as structured documents allow the separation of the structure of a document from its content and presentation, style sheets in the same way, allow a separation of the content of a document from its presentation: layout and style properties of documents are considered separately from the structuring of a document into headings and paragraphs. Thus, the way documents look changes simply by modifying just the style sheet; no change is required in the document itself to improve its appearance.

2.2 Style Specification

We shall now intentionally limit our investigation to structured documents and consider traditional print-based typography [19]. We therefore assume the following hypotheses:

- Structured documents are static documents, composed of texts and images (graphics or photos)[1].
- A style sheet is a generic style specification used to define the presentation of documents when their structure is consistent with a structure model such as a "Document Type Definition" (DTD) in SGML [8].

[1] Recent evolution of style sheets for presenting non-visual media [15], animation, synchronization [11] [12], and more generally temporal dimensions [13] [21] of documents on the internet is not considered.

– The formatting model is based on boxes [10], as in the models defined in DSSSL or CSS: any box can be defined by spatial properties and style properties: its dimensions, its position, its optional surrounding border and margin, and the style of its text content. Boxes can be embedded in other boxes.

Style specifications may be defined either using declarative languages such as CSS for HTML [18] documents in the World-Wide Web, or P, the style language of the Thot editor, or as functional specifications, as provided by the ISO standard for associating processing with SGML documents. DSSSL combines two languages: functional specifications of tree transformation and declarative style specifications. Whatever the style language, three kinds of properties may be distinguished:

Spatial properties define the dimensions and position of boxes in terms of either internal dimensions, distance, or alignment properties between boxes. So, the spatial properties are defined either by an absolute value or by a relative value.

Typographic properties concern the aspect of the text (fonts, character style, color, background, ...) or its layout (character size, spacing, ...).

Decorative properties concern boxes which are created either for ornamental purposes or for a better control of page layout (column, header, ...).

2.3 Design Approaches

Document style design is usually achieved through direct manipulation of documents. For instance, most interactive electronic publishing systems provide a modifying command for the style of text or paragraphs: after selecting a paragraph, the user chooses a command in a style modification menu (font, color, ...) for the paragraph.

Some systems use inferencing in a very simple way. As an example, by memorising the previous transformation or by looking at the first paragraph in the selection, such systems guess that the user wants the same transformation for new objects (cf. *MacDraw*). *Microsoft Word* uses the number formatting command to determine what the numbers at the beginning of all the paragraphs should be and will renumber each level appropriately, but it will get it wrong in some cases, for instance changing brackets into periods in the inner sections.

Now, most electronic publishing systems also provide facilities for a graphical layout design. However the layout design and the style specification are often considered separately (cf. *FrameMaker* [5]). Because a simple document model is considered in these systems – a document is considered as a simple list of paragraphs – the style specification is only based on paragraph and character formats.

On the other hand, some new editing environments (developed for editing HTML documents) now allow style sheet editing, providing facilities for creating or updating style sheets using both programming techniques and direct manipulation. Amaya [17], for example, provides a user interface for editing style sheets (CSS1): a CSS selector enables the user to select CSS from files, two rule selectors allow rules to be copied from one file to another. The content of a style sheet may also be edited. Indeed, the user is supposed to know to program using the CSS language as well as HTML which is not as easy as it looks.

The approach of "Design by Example" as proposed by A. Brüggemann-Klein and S. Hermann [3] in the system *Designer* is related to the direct manipulation approach that we propose. In both approaches, the system produces style specifications by inferring rules on how each document of the same type is to be formatted: in the first case, the graphic artist specifies the layout of a small number of sample documents, and a series of layout objects are implemented. In our approach, the direct manipulation interface allows a visual approach while editing any formatted document. In both approaches, the design system deals on the one hand with the formatted document, and on the other hand with the generic style specification.

3 Direct Style Design

We suggest the application of a programming by example method to graphical editing of generic style sheets: the user can display within a formatted representation of the document an example of what the style sheet will be able to produce automatically. This approach allows an interactive visual checking on any formatted document thus ensuring that the abstract style specification produces the accurate design.

3.1 The Use of a Structure-driven Editor

Our method for designing generic style sheets for structured documents makes use of an editor-formatter environment. Instead of building an environment that allows direct style sheet manipulation from scratch, we propose extending an existing editor-formatter, such as Thot. Thot is an interactive structure-driven editor-formatter with two major characteristics:

- A grammar specifies the structure of a type of document and the editor relies on this specification to produce documents with a structure consistent with this specification.
- A style sheet specifies the layout of a type of document and the formatter uses this style specification to produce formatted documents with a layout consistent with this style specification.

Furthermore, the formatted document may result both from a **generic style** sheet and from a **specific style** specification: a specific style may be embedded in the document, allowing the user to change the aspect of text or graphical element (style or formatting) and to modify the geometric position and dimension of elements – even if they contradict the generic rules as defined by the style sheet for this type of element.

By interactive editing, the user directly edits a formatted representation of the document; only actions which are consistent with the structural model (generic structure) being possible. The automatically generated presentation can therefore be changed, either globally by changing the style sheet (generic style), or locally by giving a *specific presentation* to an element of the document. However, changing the style sheet has not been possible so far by editing with Thot. Therefore, developing a style editor based on Thot will make this possible while transforming style editing commands into generic style editing commands.

3.2 From Specific to Generic Style Editing – An Example

Transforming editing actions which affect the specific presentation of a selected element into generic actions which may affect a lot of elements depending on the logical context is by no means easy, as shown in the following example.

Most editors include style editing commands enabling the updating of specific style rules. For instance, some style editing commands may be used to modify the character size of a selected *Section*: by reducing the character size from 12 pt to 10 pt, the *Paragraphs* included in this selected *Section* will be displayed with 10 pt characters. This achieves an update of the "specific rule" associated with this logical element (*Section*) within the document. This will have no effect on the style sheet, nor on the character size of any other *Section* within the document.

Conversely, the user of a generic style editor may require the style editor to extend this new character size value to each *Section* within the document. In this case, the character size rule associated with *Section* within the current style sheet should be updated, achieving an update of what is called the "generic rule" associated with the generic element type *Section*.

The problem becomes more complex when some rules are contextual. In this case, the effect of the generic style update may be limited to a context. Such context may be defined by a structural condition. If, for instance, the selected *Section* is not at the upper level (i.e. embedded in an other *Section*), the character size rule associated with *Section* may be transformed into a conditional rule so as to apply this new value of character size only to a *Section* at this level. In this case, a direct style rule must be transformed into a conditional style rule within the style sheet, as follows:

IF Section IN Section THEN character-size = 10 pt ELSE character-size = 12 pt.

A more complex situation may occur if the initial style value is defined as a relative value. For instance, if the character size of *Section* is defined as identical to the character size of *Abstract*, or conversely. The update of the rule associated with *Section* may or may not change this relationship. In such cases, the transformation must be performed with added indications from the user, who should indicate his preference when several transformations are possible.

3.3 Generic Style Transformation Process

As shown in the previous example, inferring a generic presentation from specific requirements may depend on the logical context, the generic structure, the generic style of the document, or some preferences indicated by the user.

So, let us first consider that a simple style editing action consists of:

- selecting an "*element*-box" within a formatted source document,
- editing a style property associated with an *element*-box while proposing a new value for this style property,
- validating the accuracy of the transformation.

Changes in a style sheet can then be prepared in two steps. The first step consists of analyzing the style editing command: the logical context associated with the selected *element*-box, the style rules involved in the creation of this *element*-box, and the changes required.

The second step consists of calculating the possible transformations of the style sheet which are consistent with the initial situation: a generalization process assumes the implementation of complex consistency checking mechanisms on the structural layer, in order to propose accurate generalizations and to produce the required visual effects.

After validation by the user, all the visual effects are transformed into the appropriate style sheet each time a style property is modified. The style update process produces this new style sheet. The source document is immediately formatted with this style sheet.

4 Presentation Graphs

Most of the difficulties presented above relate to the same basic problem: transforming specific rules into generic rules. In this section, we shall explain how representing a style sheet as a dependency graph, called a presentation graph, may help in detecting and checking changes for accurate generalization within a structured document.

4.1 Building a Presentation Graph

A style sheet can be considered as a series of statements in which the structural element types of the document are associated with formatting objects. For example, "paragraphs" are associated with fonts, colors, and other typographic effects. Spatial properties describe how the associated boxes will be positioned, or dimensioned. Thus, while considering a set of style properties and values associated with a selector [14], style sheets may be represented by graphs, as follows:

- Each **node** is a tuple <TYPE, STYLE> where TYPE is a structural type of element, and STYLE is a style property. One or several VAL attributes may be associated with a node. Each **attribute** represents a possible value of this style property for the structural element of type TYPE.
- Each node is connected to related nodes with a **directed arc**. The target nodes (t) are nodes on which the property value of the current node depends. The original nodes (o) are nodes that inherit from the property of another node.
- Each arc is **labelled**. The label contains a computed function (COMP) and a structural condition (COND).

Building such a graph from the style specification requires a knowledge of the generic structure in order to integrate inheritance of properties from the logical structure. Let us consider, for instance, the generic structure as described in fig. 1, and the simple style specification of fig. 2.

The presentation graph (see fig. 3) resulting from this style specification when applied to this generic structure shows that the `color` property of a *Paragraph* depends on the logical context: the node <*Paragraph*, `color`> is connected to the nodes <*Abstract*, `color`> and <*Section*, `color`>. Three attributes are associated with the node <*Paragraph*, `color`>: red, green, and black. The labels on the arcs describe

how the value of the property is calculated and which logical condition applies to it: the `color` of a *Paragraph* is either red in the *Abstract*, or green in the *Appendix*, or black elsewhere. For instance, arc (a) in fig. 3 which connects node <*Paragraph*,`color`> to node <*Abstract*,`color`> is labelled by the following computed function:

COMP = [VAL(o) = VAL(t)]

and by the following structural condition:

COND = [*Paragraph* = In (*Abstract*)].

```
Report = (Title, Abstract, Contents, Appendix)
Abstract = (Paragraph)*
Contents = (Section)*
Section = (Section_title, (Paragraph)*)
Appendix = (Appendix_title, (Section)*)
```

Fig. 1. A generic structure

```
(1)   Report {color: black}
(2)   Abstract {color: red}
(3)   Appendix {color: green}
(4)   Section_title {color: blue}
```

Fig. 2. A style specification

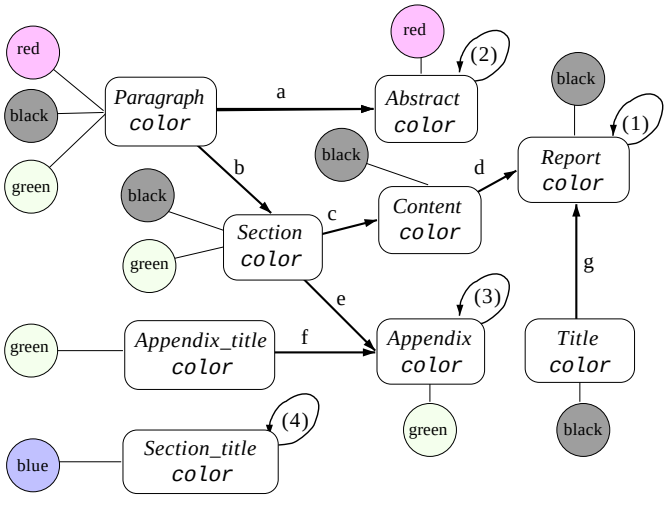

Fig. 3. A presentation graph

4.2 Transformations on the Presentation Graph

Several areas may be identified in a presentation graph to check the "influence" of a style rule. Consider a style property P, and a selected element E of type T. The logical context of E is supposed to be determined by the embedding elements of E. Consider the sub-graph **G** of the presentation graph only describing property P, and call node N = <T, P>: the influence areas of (P, E) are then defined as follows within the sub-graph **G**:

1. the **style path** associated with E contains the node N = <T, P> and all nodes connected to N with an arc whose logical condition is satisfied by the context of E. This path helps to detect all the style rules which have an effect on the style of E.
2. the sub-graph $\mathbf{G}^-$: contains all the nodes which are the origin of an arc directed to N. This area covers all the style rules which may be involved by a modification of P on N, *i.e* elements which may inherit from the value of the P property of the E element.
3. the sub-graph $\mathbf{G}^+$: contains all nodes which are the target of an arc originating from N (directly or indirectly). This area covers all the style rules which must be reconsidered when modifying or deleting N or arcs originating from N.

As an example, consider the presentation graph (fig. 3), and the new value "blue" for the `color` of a *Section* element E within the *Content* of a *Report*. The style path associated with E is composed of (c) + (d) and goes from the node <*Section*, `color`> to the node <*Report*,`color`>. $\mathbf{G}^-$ includes the node <*Paragraph*, `color`>. $\mathbf{G}^+$ includes the node <*Appendix*, `color`>.

The generalization process examines the label of arc (e) to $\mathbf{G}^+$ and arc (c) in the style path to calculate possible updates in **G**. A first update may consist of creating a direct rule on the node <*Content*, `color`> setting the color to blue, deleting (d), and replacing the black attribute on <*Content*, `color`> by a blue one. In this case, only *Sections* included in the *Content* of a *Report* will be blue. *Sections* which are included in the *Appendix* will remain green.

A second update may consist of deleting (c) and (e), and creating a direct rule on the node <*Section*, `color`> setting the color to blue so that any section color will be blue whatever the context, and replacing the green attribute on <*Section*, `color`> by a blue one. In any case, the black attribute on <*Section*, `color`> will be deleted.

4.3 Advantages and Limits of the Presentation Graph

The presentation graph matches the structural context in so as far as the contextual rules rely upon inheritance from the logical structure. Conditional rules which involve several types are not represented in this graph. Only values related to adjacent nodes are described. However, this is a natural way of specifying a generic style while using logical inheritance and inclusion relationships. And it is suitable for supporting most of the rules as defined in CSS1 where relative rules only depend on the parent element.

Indeed, the graph helps to clarify style specifications in many ways: it can be used to reduce the style sheet by grouping style definitions at the highest level possible. Thus,

only efficient style rules are kept in the style sheet when its construction is based on such a graph. This application offers a number of additional advantages over style validation which are not presented here.

5 P-edit, Current State of Development

P-edit is a prototype of a generic style editor. Thot is the structured editor which we have taken as a basis to build this generic style editor. This propotype enables the updating of style sheets which are written in the P language, the style sheet language used by Thot. It is developed through the application programming interface (API) provided by Thot. The interactive formatter – integrated within Thot – applies the style specification as defined in the P model in the source document. Initial implementation is limited to typographic and spatial properties of elements and validates the presentation graph.

5.1 User Interaction

As described in section 3.1 (and in the Thot Manual [2]), the *specific presentation* allows the user to change the aspect of text or graphical elements, and the color or the formatting of the elements of a document. The geometric position and dimensions of elements can be changed using the mouse to move or resize a box, and color can be modified by selecting within a color palette, etc. In addition to these commands, the following commands are provided in P-edit to edit generic style sheets:

Control of the style editor Starting the style editing session: this transforms all the subsequent specific style editing actions into generic style editing actions and some new style editing actions become available.
Creating a new style sheet and stopping the style editing session: the P compiler produces a new style sheet which is immediately used to format the document[2].

New style editing actions Creating and editing decorative boxes: decorative boxes are temporarily considered as element-boxes within the source document. This result is obtained by adding/removing extension into/from the structure model.
Editing numbering properties: all the possibilities for the style numbering of a list element are provided. For visual checking, the changes made on the style numbering immediately apply to the selected list.

Generalization Designating related elements: either for calculating values or for conditional application.
Choosing a method to calculate relative style properties: with a definite increment and related to a selected element.
Choosing a generic application: several inferences are provided, and some may be updated or completed by designing related elements.

Box visualization For better visual editing, outline or colored background of boxes are provided. Changing visibility enables the display and selection of embedded boxes when bounding boxes overlap.

[2] An initial style sheet is assumed to be provided. Our intent is to add facilities within P-edit to build a very simple style sheet inferred from the structure model.

5.2 First Experiments

As shown by the user interface (fig. 4), the user does not edit the style sheet, but changes the appearance of the source document in two steps: the first step consists of using the style dialog box which describes the style properties, the numbering properties, ..., or the colors of the currently selected element, or resizing the box associated with the selected element, to update a style property. The second step consists of choosing the scope of the modification, assisted by the generalization process, before asking the style update process to apply the generic modification to the current style sheet. ¿From our first experiments with P-edit, we can outline some points:

- while providing appropriate feedback about what the style editor is doing, this method (direct manipulation, design by example, visual control) insures accuracy and full specification of the style for the source document,
- while applying the same style sheet to any document belonging to the same class, visual checking is easy and further updating of the style sheet is possible. However, this method does not ensure that any document of the same class will look as aesthetic as defined in the source document: some elements may be missing, thus changing the look-and-feel of the layout.
- this way of producing style sheets for structured documents supposes that the user is aware of the document structure. Otherwise, he/she may have difficulty in selecting the appropriate element before editing a style property or selecting a related element to define logical contextual dependencies. Some visualization of this structure and of the presentation graph would provide the user with a global vision of the dependencies and should be further investigated.

6 Conclusion

This paper has presented a tool for the design of style sheets for structured documents, which does not require users to program the style language. The proposed approach is based on a presentation graph and enables the generalization process to produce the style sheet properly. In this approach, designers or authors directly modify the presentation on a formatted document. By immediately visualizing the formatted document, users are able to discover relationships among logical components and style rules. This ability helps users to update the generic style in an efficient way.

A first prototype, based on the structured Thot editor, provides an interactive environment for editing the typography, topology, and style of a document, and for updating style sheets written in the P language. Since P-edit is a prototype used for the validation of our generic approach, it does not provide all the functionality of a full system. Some features such as page layout, footnotes, views, or decorative boxes should be improved. And we have made the simplifying assumption that an initial style sheet is provided.

Until now, not much attention has been paid to the problem of generic style editing. The results of this experiment reveal that this approach can be used to edit style sheets with simple contextual conditions as defined in the CSS language.

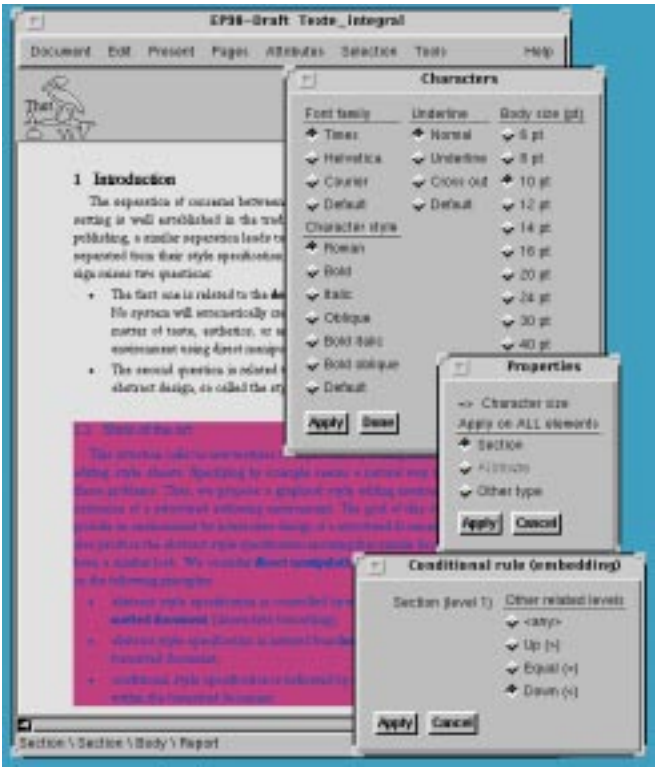

Fig. 4. Character style editing of a *Section* with P-edit

In conclusion, this experimental style editor is a first step in improving the typography of electronic documents. Further researches should explore macro-typography knowledge with a view to integrating the expertise of professional designers or typographers and so producing better-quality style sheets.

Acknowledgements

We would like to thank anonymous rewievers for valuable comments on this manuscript and C. Hérault for her help in bringing this about. We also thank Centre de Formation des Traducteurs, Terminologues et Rédacteurs (University of Rennes 2) and Heather Brown for their proof reading.

References

1. S. Adler, A. Berglund, J. Clark, I. Cseri, P. Grosso, J. Marsh, G. Nicol, J. Paoli, D. Schach, H.S. Thompson, C. Wilson, *A Proposal for XSL*, Submitted to W3C, http://www.w3.org/TR/NOTE-XSL, 27 August 1997

2. S. Bonhomme, V. Quint, H. Richy, C. Roisin, I. Vatton, *The Thot User's Manual*, Inria-Imag, http://opera.inrialpes.fr/doc/thot/Thotman-E.html, 1997.
3. A. Brüggemann-Klein, S. Hermann, "Design by example: A user-centered approach to the specification of document layout", *Proc. of ICCC/IFIP Conference on Electronic Publishing'97: New Models and Opportunities*, F. Rowland, J. Meadows ed., ICCC Press, Washington, DC, pp. 223-236, 1997.
4. A. Cypher ed., *Watch What I Do: Programming by Demonstration*, MIT Press, Cambridge, MA, 1993.
5. FrameMaker, *FrameMaker User Manual*, Frame Technology Corporation, San Jose, CA, 1995.
6. R. Furuta, V. Quint, J. André, "Interactively Editing Structured Documents", *Electronic Publishing*, vol. 1, num. 1, pp. 20-44, April 1988.
7. C. Hüser, W. Möhr, V. Quint ed., *Electronic Publishing, Document Manipulation and Typography*, Proc. of the Fifth International Conference, EP-odd, 6(4), December 1993.
8. ISO, *Information processing - Text and Office systems - Standard Generalized Markup Language (SGML)*, ISO 8879, October 1986.
9. ISO, *Information techmology - Text and Office systems - Document Style Semantics and Specification Language (DSSSL)*, ISO/IEC DIS 10179, 1996.
10. D.E. Knuth, *The TEXbook*, Addison Wesley Publishing Company, Reading, MA, 1988.
11. M. Jourdan, C. Roisin, L. Tardif "Édition et Visualisation Interactive de Documents Multimedia", *Proc. of Electronic Publishing 1998*, J. André, H. Brown, ed., Springer-Verlag, 1998.
12. P. King, H. Cameron, H. Bowman, S. Thompson "Synchronization in Multimedia documents", *Proc. of Electronic Publishing 1998*, J. André, H. Brown, ed., Springer-Verlag, 1998.
13. N. Layaïda, L. Sabry-Ismail, "Maintaining Temporal Consistency of Multimedia Documents Using Constraint Networks", *Multimedia Computing and Networking 1996*, M. Freeman, P. Jardetzky, H.M. Vin, ed., pp. 124-135, SPIE 2667, http://opera.inrialpes.fr/OPERA/BibOpera.html, January 1996.
14. H.W. Lie, B. Bos, *Cascading Style Sheets, level 1*, W3C Recommendation, http://www.w3.org/TR/REC-CSS1, 17 December 1996.
15. C. Lilley, T.V. Raman, *Aural Cascading Style Sheets (ACSS)*, W3C Working Draft, http://www.w3.org/TR/WD-acss, 28 March 1997.
16. B.A. Myers, "Demonstrational Interfaces: A Step Beyond Direct Manipulation", *IEEE Computer*, vol. 25, num. 8, pp. 61-73, August 1992.
17. V. Quint, C. Roisin, I. Vatton, "A structured Authoring Environment for the World-Wide Web", *Computer Networks and ISDN Systems*, vol. 27, num. 6, pp. 831-840, April 1995.
18. D. Raggett, *HTML3.2 Reference Specification*, W3C Recommendation, http://www.w3.org/TR/REC-html32, 14 January 1997.
19. H. Richy, C. Hérault, J. André, "Notion de feuille de style", *Cahiers GUTenberg*, vol. 21, pp. 127-134, http://www.univ-rennes1.fr/pub/gut/publications, June 1995.
20. R. Southall, "Presentation Rules and Rules of Composition in the Formatting of Complex Text", *Proc. of Electronic Publishing 1992*, C. Vanoirbeek, G. Coray, ed., pp. 275-290, Cambridge University Press, Cambridge, UK, 1992.
21. L. Weitzman, K. Wittenburg, "Automatic Presentation of Multimedia Documents Using Relational Grammars", *Proc. of the Second ACM International Conference on Multimedia*, pp. 443-451, San Francisco, CA, October 1994.