BIBLE: a system for Design and Management of Context-Controlled Documents

Claude Pasquier

BIBLE : a system for design and Management of Context-controlled Documents

Claude Pasquier

Informatique CDC, Recherche Développement Techniques Avancées
4 rue Berthollet BP 16 - 94114 Arcueil Cedex, France.

I3S - CNRS - UNSA
Rue Albert Einstein - BP 145 - 06903 Sophia-Antipolis Cédex, France.

ABSTRACT : International standards for the representation of structured documents like ODA [ISO8613 89] or SGML [ISO8679 86] are well adapted for the design and the generation of long and sophisticated documents like books or technical documentation. But, in the tertiary industry, most documents are intended for clients. Their constitution depends on the client profile. For these documents, the traditional document processing model (figure 1) is inappropriate, it is necessary to introduce an intermediate step between editing and formatting. This step permits choice of content and structure based on the client profile. We call this new phase the automatic design (figure 2). Between this new phase and the editing process, we need a specific logical structure with variable parts. We use the term context-controlled to express that the description of the document depends on external information coming from the client profile.
A structured document is described by a fixed indiscriminate logic of subordination. To represent a variable subordination, a context-controlled document also contains instructions on how to use variables from the client profile. Some of these instructions are specific to one document, but other concern a type of document. So, they can be specified in the document model.
All models are gathered in a document database. Modularity allows us to create new models by reusing existing modules. But, this concept doesn't provide for modification of shared module neither does it introduce a sufficient structure for modelling large numbers of models. We have resolved this problem by using inheritance. This concept permits us to structure the document database in an tree like manner. Both the modelling of the document database and the working of editors are described by using an object-oriented approach.

KEYWORDS : structured document, context-controlled document, variable document, automatic design, client-selective mailing, inheritance, reusability.

Introduction

International standards for the representation of structured documents like ODA [ISO8613 89] or SGML [ISO8679 86] are well adapted for the design and the generation of long and sophisticated documents like books or technical documentation. But, in the tertiary industry, most documents are intended for clients. Their constitution depends on the client profile.

By the term client profile, we mean all information which may be used to govern the choice of the structure, the content or the presentation of a document. This information can refer either the recipient of the document, its writer or a particular context like economic position.

For high volume client-selective documents, the traditional document processing model (figure 1) is inappropriate. Volumes are such that cost prohibits the creation of a specific document for each person who receives a mailing, yet the same document may not be applicable for each targeted reader.

Moreover, current standards don't include the concepts of reusability, modularity [Quint 89] or inheritance, even though they have an object-oriented approach. But these concepts are necessary to modelize more than one kind of document.

The group Caisse des Dépots et Consignations is especially concerned by these problems since thousands of documents are produced every day. Therefore, a project named BIBLE was launched in 1990, to define a new architecture for the production of documents.

In the first section of this paper, we explain the differences between a traditional document processing model and the processing model used in BIBLE which permits the semi-automatic design of documents. We introduce in this section a new kind of document representation called context-controlled document.

In the second section we describe context-controlled documents.

In the third section, we discusse the creation and modification of document models which are used to guide the editing and formatting of documents..

Finally, we introduce in the fourth section, the concept of inheritance in order to provide commun structure for all document models.

1 A new document processing model

In traditional systems for the manipulation of structured documents, a document is represented by a logical structure created in accordance with specifications hold in a generic structure (called DTD in SGML). The layout structure of the document is generated by a formatting process from the logical structure and some presentation rules. The set of these rules is also called generic layout structure. After that, an imaging process produces the final image of document adapted to the output medium (figure 1).

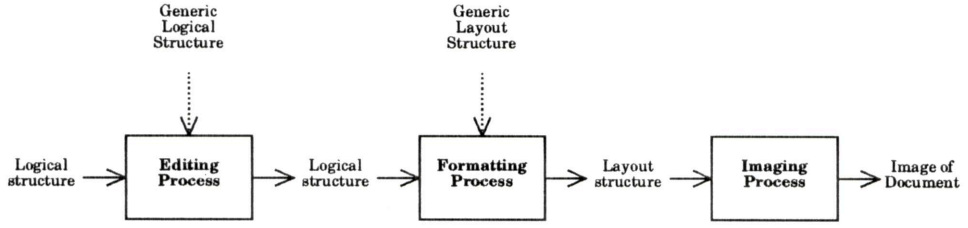

Figure 1 : A traditional document processing model

To automate high-volume client selective mailing, it is necessary to introduce an intermediate step between editing and formatting. This step permits choice of content and structure based on the client profile. We call this new phase the automatic design (figure 2).

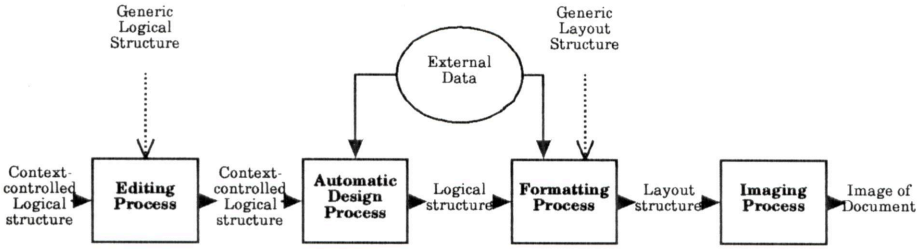

Figure 2 : The model of document processing in BIBLE

Between this new phase and the editing process, we need a specific logical structure with variable parts. The choice of document parts is governed by data issuing from client profile. We use the term context-controlled to express that the description of the document depends on external information coming from the client profile. Therefore the editing process must allow the user to work on variable parts of the context-controlled logical structure. This structure is also called a context-controlled document.

2 The context-controlled document

The tree-like structure of a document is represented by a list of expressions describing the content of each object. For example,

Body = {*Paragraph#1 Paragraph#2 Paragraph#3*}

Means that the object named *Body* contains the tree paragraphs between the brackets.

In this representation, documents are described by a fixed indiscriminate logic of subordination.

2.1 Variable subordination

A context-controlled document must also contain instructions on how to use variables from the client profile in order to automatically design documents. These instructions may be very complicated because the client profile is used in different ways :

- It may be printed as is. For instance, the name and the address of the recipient of a letter is simply copied from the database to the document.
- It may be used to choose content. For example, the form of polite address is computed according to the sex and the family condition of the recipient.

- It may be used to govern both structure and content. For instance in an insurance contract, some clauses or chapters can be determinated in accordance with client characteristics.

2.2 Attaining variable subordination

The use of external data is denoted by *$ <var >*. Like in the sentence :

"We let you know that your banking account number $BankAccNumber is in debit balance of $BankAccBalance ECUs".

In addition, to express control structures in the context-controlled document, two other constructors are used :

- **The choice** between several elements, denoted by the keywords *CASE $ <var > OF*.

```
e.g.: FormOfAddress = CASE $sex OF {
                      "Male" : "Mr"
                      "Female"      : CASE $married OF {
                                      "Yes"  : "Mrs"
                                      "No"   : "Ms"
                                      }
                      }
```

- **The repetition** of an element, expressed by the construct *LIST (<lowerLimit >, <upperLimit >) OF*. *LowerLimit* and *upperLimit* are either a constant or a variable.

```
e.g.: BranchsReports <-- LIST (1 .. $nbBranch) OF BranchReport
```

2.3 Linking to the database

To generate structured documents from a context-controlled document, we need to make a link between variables used in the document and data issuing from a database. This is made with the command :

```
GENERATE <Context Controlled Document >
        (<list of variables used in documents >)
    USING <list of variables from database >
```

In combination with a database language like SQL [ANSI 86], this allows us to select the documents to produce. For instance, to send a follow-up letter to all late-payers :

```
GENERATE LatePayerFollowupLetter ($name $address $date
$bankAccNumber $bankAccBalance)
    USING SELECT c.name c.address p.date c.bankAccNumber
c.bankAccBalance
    FROM (Client c, Parameters p)
    WHERE c.bankAccBalance < 0
```

3 The design of document models

Models are used in the editing process and the formatting process. They are created and modified by a model editor.

3.1 The logical models

The control structures in context-controlled documents use the data from the client profile. Some of these control structures are specific to one document. They are mentioned by the user in the editing process. But other control structures concern a type of document. For instance, the determination of the polite address is the same for all letters. So it can be specified in the letter model. The same approach applies to the name and address of the receiver or the date.

When a document is created, control structures are simply transmitted from the model to the document. In this way, the writer can focus his full attention on the essential of a document : Its content :

```
Letter = {
   Header = {
      Name
      Address = $address
      Date = $date}
   Body}
Body = {
   LIST [1..*] OF
      Paragraph = CHOICE {TEXT TABLE}
      Signature = CHOICE {
            $bigboss-signature
            $staffManager-signature
            $chiefAccountants-signature}}
Name = {
   CASE $sex OF {
      "Male" : "Mr"
      "Female" : CASE $married OF {
                        "Yes"  : "Mrs"
                        "No": "Ms"}}
        $name}
```

In this example, the structure of a document is expressed by the grammar [Furuta 88] [Ingold 88] which uses the following constructors [Quint 88] :

- **The aggregate**, represented by a list of elements delimited with brackets.
  e.g : Letter = {Header Body}
- **The choice** between several elements denoted by the keyword *CHOICE*.
  e.g : Paragraph = CHOICE {TEXT GEOMETRICGRAPHIC}
- **The repetition** of an element, expressed by the construct *LIST [<lowerLimit> .. <upperLimit>] OF*.
  e.g : Body = LIST [1..] OF Paragraph*
  The star denotes an unspecified limit.

The absence of an element is represented by the *NULL* constant. The content of some elements can also be specified in the grammar.

The document model of *Letter* is expressed in modules. *Body* and *Name* are indeed described outside the model. In this way they constitute independants structures which can be reused in the description of other documents.

3.2 The layout models

The image of a document is built up from the logical model, with the help of presentation rules [Quint 88]. These rules can be associated with each logical object, or better still, be described separately. With the latter solution, we can have different presentations for the same logical model : a screen layout and a paper layout for example.

From the layout point of view, a document is simply a gathering of boxes [Knuth 86]. Each box contains its own properties, like height, width, border size. The grammar of layout model specifies hierarchy between boxes.

The layout model of a document can be described in a modular way, like the logical model. However, this is not sufficient to allow us to reuse boxes. We must indeed take their position into account.

In the ODA standard, box position is specified in the box itself with the attribute *Position* (ISO8613-2 p.65) which prohibits reuse of the box. To solve this problem, we mention positions in the grammar, when boxes are used. In this way, box are context independant and can be used in several documents at different places.

This allows us easily to build layout models of documents by placing boxes on the page.

The position and dimension of boxes is expressed either by fixed coordinates or with a set of constraints [Franchi 89] such as separation from other boxes, alignment, dimension suitable for content, etc. For instance, the layout model of a letter can be described by :

```
Letter-lay = {
  FirstPage = {
    AT <POS> FirstPageHeaderBox = {
      AT <POS>NameBox
      AT <POS>AddressBox
      AT <POS>DateBox
      }
    AT <POS>FirstPageBodyBox}
  OtherPages = {
    AT <POS>OtherPagesHeaderBox = {
      AT <POS>DateBox2}
    AT <POS>OtherPagesBodyBox}}
```

3.3 Linking logical models with layout models

In order to be complete, the description of a model must contain links between its logical and layout aspects.

In the project BIBLE, we have chosen to express relations separately, in the form of a set of links :

Name --> NameBox
Address --> AddressBox
Date --> DateBox, DateBox2
Body --> FirstPageBodyBox:OtherPagesBodyBox

This example shows the tree different possibilities :

- *Address --> AddressBox* means that the logical object *Address* must be displayed in the box named *AddressBox*. If *AddressBox* can't hold the content of *Address* (after reshaping if the box dimension is variable) then the content of *Address* is truncated.
- *Date --> DateBox, DateBox2* means that the logical object *Date* must be displayed in *DateBox* **and** in *DateBox2*. As for *Address* if a box can't hold the entire *Date*, it is truncated.
- *Body --> FirstPageBodyBox:OtherPagesBodyBox* means that the content of *Body* is displayed in *FirstPageBodyBox*. If *Body* can't fit in *FirstPageBodyBox*, then the rest of the content is displayed on *OtherPagesBodyBox*.

4 The management of several models

All models are gathered in a document database. Modularity allows us to create new models by reusing existing modules. But this concept doesn't provide for modification of shared modules in a document neither does it introduce a sufficient structure for modelling large numbers of models.

4.1 The limits of reusability

In section 3.1 we said that modularity allows us to reuse structured objects. This is true, but modularity has its limits.

Suppose that we want to create a new model of letter in which paragraphs will be grouped in sections.

To create the structure of *NewLetter* we can reuse *Header* of *Letter*. For describing the *Body*, we have two solutions : either we rewrite it entirely, or we copy and modify *Body* from *Letter*.

In the two cases, the result is the same : *Body* of *Letter* and *Body* of *NewLetter* are described in an independent way. We lack all advantages of reusability. For instance, if we latter want to modify all letters in order to put the writer's name under his signature, we need to modify more than one model.

4.2 The use of inheritance

We have solved this problem by using inheritance. The structure of *NewLetter* can indeed be expressed in terms of modifications of the model of *Letter*. In this

way, all modifications done within the structure *Letter* will be incorporated automatically in the structure of *NewLetter*. So to add the name of the writer on all letters, we only have to modify a single model.

In BIBLE, we have chosen to generalize this concept by decreeing that each document be created by specialization. BIBLE defines a single, root document represented by an empty structure. Thus all documents are structured in an tree like manner (Fig. 3).

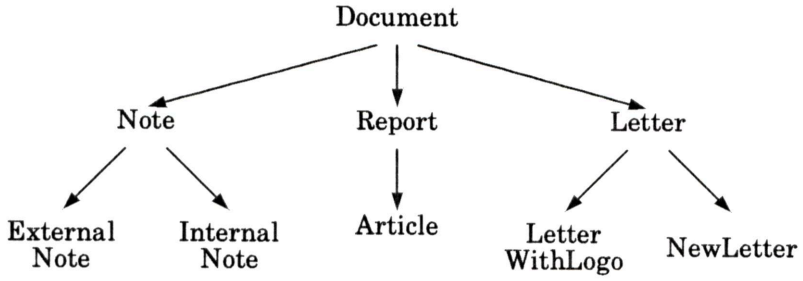

Figure 3 : Example of a model hierarchy

4.3 Using an object-oriented approach

BIBLE uses the concepts, of instanciation and inheritance of object-oriented programming field. Both the modelling of the document database and the functioning of editors can be described by using an object-oriented approach.

The structure of each model can be gathered in a class. Thus the context-controlled documents generated from models are described by instances of classes. As in all object-oriented languages we describe the behaviour of instances in classes and the behaviour of classes in metaclasses [Murata 89].

We describe the functions of our syntax-directed document editor in classes and the working of our model editor in metaclasses.

Summary

This article is primarily concerned with the generation of documents from context-controlled documents and the design of document models. The weakness of this part is the use of a set of links to describe the matching between the logical and the layout aspect of a document. We are trying to replace this system by a dynamic matching between the two structures. We are also working on the definition of a new structure, that we'll call context-controlled layout structure, to allow a variable control of the document's presentation

However the most innovative aspect of our project concerns the organization of a set of models.

At the present time, we are defining and experimenting the concept of inheritance between structured objects. We are studying the possibility of using

multiple inheritance. For instance, creating a new model of a letter which inherits both the *Logo* defined in *LetterWithLogo* and the *Section* used in *NewLetter* (see section 4.1 & 4.2).

Acknowledgements

I am grateful to the members of Reaseach Department of Informatique CDC who kindly read primary versions of this paper and made many constructive comments. I also wish to thanks Prof. Paul Franchi-Zannettacci for his valuable advices and Mr. Kent Hudson for his help.

References

[ANSI 86] American National Standard for Information Systems, *Database Language SQL*, ANSI X3.135-1986, October 1986.

[Franchi 89] P. Franchi-Zannettacci, *Context-Sensitive Semantics as a Basis for Processing Structured Documents*, WOODMAN'89

[Furuta 88] R. Furuta, V. Quint, J. André, *Interactively editing structured documents*, Electronic Publishing, vol. 1, no 1, pages 19-44, April 1988.

[Ingold 88] R. Ingold, R. Bonvin, G. Coray, *Structure recognition of printed documents*, in Document Manipulation and Typography, Proceedings of the International Conference on Electronic Publishing, Nice, April 1988.

[ISO8613 89] ISO 8613, *Information Processing - Text and Office Systems - Office Document Architecture (ODA) and Interchange Format*, 1989.

[ISO8879 86] ISO 8879, *Text and Office Systems - Standard Generalized Markup Language (SGML)*, 1986.

[Knuth 86] D. E. Knuth, *The TEXbook*, Addison-Wesley Publishing Company, 1986

[Murata 89] M. Murata, *An object-oriented interpretation of ODA*, WOODMAN'89.

[Quint 88] V. Quint, *Systems for the manipulation of Structured Documents*, Cambridge University Press, in Structured Documents, pages 39-74, 1989.

[Quint 89] V. Quint, I. Vatton, *Modularity in structured documents*, WOODMAN'89.