Advanced Modeling and Browsing of Technical Documents

François Bry
Institute of Computer Science
Oettingenstrasse 67
80538 Munich, Germany
Francois.Bry@informatik.uni-muenchen.de

Michael Kraus
Institute of Computer Science
Oettingenstrasse 67
80538 Munich, Germany
Michael.Kraus@informatik.uni-muenchen.de

ABSTRACT

This article proposes three techniques for improving the modeling and browsing of electronic books for technical documents specified in XML. Technical documents are for two reasons prime applications for electronic books. First, electronic devices are nowadays widely used for the distribution, storage, and updating of such documents. Second, since most technical documents have rather complex semantics and structure, they might be easier to consult as electronic books with their advanced functionalities than as paper prints.

The techniques proposed in this article are as follows. First, *author's views* are proposed as a means for grouping semantically related, yet different contents. Second, *browsing style sheets* are proposed for distinguishing between the semantics of hypertext links and their traversal behavior more drastically than it is the case with current XML hypertext links. Third, a tool called *Reader's View* is described with which a reader can annotate and restructure an electronic book while exploring it. These techniques remind of former proposals made in the hypertext community. The focus of the present proposal is the smooth integration of the proposed techniques within the existing XML/HTML world so as to exploit existing standards and tools as far as possible.

Keywords

technical document, adaptive document, hypertext, annotation, XML

1. INTRODUCTION

Technical documents as those used in industry (e.g. maintenance instructions), in training (e.g. procedure descriptions), and in education (e.g. lecture notes) are for two reasons prime applications for electronic books. First, such technical documents are nowadays distributed and maintained more and more in electronic form. Second, because technical documents often have complex semantics and consequently a complex structure, they are often easier to consult as electronic books than as paper prints. Indeed, electronic books with their nowadays widespread browsing tools offer functionalities such as string or topic search, hypertext links, pop-up text extensions, and sometimes restructuring facilities that often considerably facilitate the consultation of complex documents. For training and educational purposes, such functionalities especially search and (in-context or out-of-context) text extensions within the same visual space have proved to be useful.

This article investigates techniques for improving the modeling of semantically complex textual contents as well as the browsing of complex document structures like those of technical documents. Three techniques are proposed: *author's views*, *browsing style sheets*, and *reader's views*.

Author's views are a means for grouping semantically related, but different contents, e.g. texts at different levels of detail. Often, the modeling of an electronic document involves semantically related, yet different textual contents. This is for example the case of an assembly procedure which can have various descriptions at different level of detail. A summarized description of the procedure might serve to time and personnel assignments. Complementing this, a detailed description of the procedure might describe the processing steps. Lecture notes also often involve complementary textual contents either at different level of detail, or made available for different purposes. A mathematical proof might e.g. be given as short teacher's notes to be used as a reminder while lecturing, or as a detailed after-lecture explanation for students. Both examples of electronic book applications have in common that complementary texts with closely related semantics, what we call *views*, are available. Commonly, views are modeled as different documents. This makes their maintenance difficult, for semantically related texts are distributed over different document structures. A grouping mechanism is proposed with which various views can be expressed and structured together within the same document. It is argued, that the capability to express such views might considerably improve the modeling of complex technical documents.

In a *browsing style sheet* attached to a hypertext document the traversal behavior of hyperlinks is specified. In the document itself, links are defined only as semantic relationships between data items. The distinction between link semantics and link traversal behavior underlying the browsing style sheets is analogous to the distinction between a generic markup language such as XML, whose purpose is to model

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
SAC 2002, Madrid, Spain
Copyright 2002 ACM 1-58113-445-2/02/03 ...$5.00.

the logical structure of data items, and a style sheet language such as CSS, DSSSL or XSLT/XSL-FO, whose purpose is to specify the layout for a given logical structure. Browsing style sheets increase the device and data independence of hypertexts. They make it easier to model electronic books to be consulted on various types of devices such as projection, workstation, and hand-held computer screens. Device independence is especially desirable for technical documents that often have to be consulted on rather different devices. The same assembly procedure might for example be displayed on a projection screen during a class and also consulted on a hand-held computer by a technician applying it.

The idea of a *reader's view* is that of a considerably enhanced and combined history and bookmark functionality of a browser together with an annotation mechanism. A browser offering this functionality makes it possible for readers to reorganize and annotate at their wills the electronic documents they are reading. This is often useful with complex texts. While reading a technical document, one often skips introductory chapters. While learning from a textbook, one often writes notes. The structure of such reader's notes often does not perfectly match with that of the textbook read.

The proposals of this article are in some aspects reminiscent of former proposals made in the hypertext community. E.g. [2] gives an overview of existing adaptive hypermedia approaches comprising adaptive presentation (this article's notion of author's views is an example for this) and adaptive navigation support (the technique of browsing style sheets can be used to accomplish this). The idea of annotations is being examined by [1], and simple history and bookmark functionalities (that this article's notion of reader's view considerably extends) are already part of every web browser. The main advantage of this article's proposals is the smooth integration of these techniques within the existing XML/HTML world so as to use existing standards and tools as far as possible. Very small extensions to existing standards and tools in order to build such funcionalities are described. Further techniques investigated in the adaptive hypertext community such as user modeling and "learning performance" are consciously not considered in this paper for they would require significant extensions to existing standards and tools.

The article consist of five sections. This introduction is followed by a section, Section 2, devoted to author's views. Section 3 introduces browsing style sheets. Section 4 is devoted to reader's views. The last section, Section 5, briefly describes a prototype implementing the functionalities described in this paper. Acknowledgments and References follow.

2. AUTHOR'S VIEWS

2.1 Modeling Author's Views

Author's views are inspired from database views. In many databases, views ensure the conceptual independence between the primary data stored in the database and the various interpretations needed for applications using the database. Common database views are partial look-ups of the data stored. Most database applications make use of views of some sort.

Views also make sense in documents, especially in text documents with complex structures and/or contents. Such documents, for example, technical manuals and textbooks, are often read at different *semantic levels*. This is for example the case of maintenance manuals that can be read by both, an engineer for estimating the time needed for performing some operation, and a worker for learning how to perform the operation. It is also the case with textbooks that can be read by a teacher preparing a lecture, or by a student learning for an examination. The two readers will probably focus on different parts. A teacher might, for example, focus on the arguments of the introductory sections and skip proofs or explanations he is already familiar with, or restrict his attention to overviews of these proofs and explanations, if such overviews are available. A student, in contrast, is likely to skip the introductory parts and to devote more attention to detailed expositions of proofs and explanations.

Modeling each view over a text as a separate document, i.e. modeling one document for each view – such as a *teacher-view* document and a *student-view* document in the abovementioned example – would have several drawbacks. This would induce redundancies in cases where distinct views coincide on some text items. Because of these redundancies, updating errors could happen. Hence, the consistency between the different texts giving complementary *views* of the same content would be difficult to maintain.

It appears desirable to group together the various views of each text item into a single document. I.e., a document with views should not be structured as a sequence of views, but instead retain the structure common to all the view fragments and group together the views corresponding to the same text items. Doing so, the resulting document consists of a sequence of `views-groups` elements, each collecting the items related to the various views.

This approach is illustrated on an example as follows. In this example, three possible views over this very article are considered:

- a *detailed* view which consists in this article,
- a *digest* view which is a shortened digest of this article,
- a *slide* view to be formated into slides for a screen presentation of this article.

```
<article>
 <title>Advanced Modeling and Browsing of
  Technical Documents</title>
 <abstract>
  <views-group>
   <view type="detailed">This article proposes
    three techniques for improving the modeling
    and browsing [...] so as to use existing
    standards and tools as far as possible.</view>
   <view type="digest">Three techniques are
    proposed for improving the modeling and
    browsing of technical documents specified in
    XML: "author's views" for grouping
    semantically related contents; "browsing style
    sheets" for distinguishing between the
    semantics and the traversal behavior of
    hypertext links; and "reader's views" using
    which a reader can annotate and restructure an
    electronic book while exploring it.</view>
   <view type="slide">Improving the modeling and
```

```
    browsing of technical documents
    <list>
     <item>Author's views</item>
     <item>Browsing style sheets</item>
     <item>Reader's views</item>
    </list></view>
  </views-group>
 </abstract>
 <section>
  <title>Introduction</title>
  <views-group>
   <view type="detailed">[...]</view>
   <view type="digest">[...]</view>
   <view type="slide">[...]</view>
  </views-group>
 </section>
 . . .
</article>
```

The elements of type `views-group` are grouping constructs for collecting together various views over a text item – such as the abstract of this article. The content of a `views-group` element consists of `view` elements. The values of the `type` attribute of a `view` element serve to name the views involved: these values can be freely defined by the XML application designers.

The overall structure of the document in the example above is that of the initial article, i.e. the present article. This structure is shared by the three views over the document. Sharing this structure contributes to facilitate the maintenance of the document. Indeed, structure changes do not have to be repeated in different view documents, what would be error prone, and a content change in one view item suggests to perform the corresponding changes in the associated other view items, a feature which could be supported by *view-aware* XML tools or editors. Such an editor could for example display all views in parallel, each in a separate window, and prompt the author after an update of one view item for possible updates of the corresponding items of the other views.

The idea of adaptive presentation in hypertexts has already been investigated in e.g. [8, 6, 7]. The main advantage of the above described approach is it simplicity with regard to its integration into the XML/HTML world.

2.2 Related Notions: Ruby and Footnotes

The `views-group` and `view` constructs illustrated above remind of the ruby elements [5] recently added to the latest specification of XHTML and CSS3. This is not by chance: Views and ruby annotations are closely related notions.

Ruby annotations, short *rubys*, are short annotations to a base text used in Japanese and Chinese documents to indicate pronunciation or for some other purpose. Ruby annotations are usually formatted alongside the base text using a smaller typeface (the name *ruby* stems from a typeface frequently used for such annotations).

A difference between views, as considered in this paper, and ruby, as specified in [5], is that views do not assume a *base text* while the specification [5] does. This difference is not significant, however, since the semantic labeling provided by the values of the type attribute can be used for marking a view as *base text*.

Another difference is that ruby annotations are in general very short, typically a few words long, while views often extend over several sentences or paragraphs. This difference, however, is a difference of usage, not of principle. Thus, views as considered in the present paper can be seen as a generalization of structural ruby annotations [5].

Views, as defined above, can also be applied to specify *structural footnotes*. While classical footnotes are connected to a position in the text they refer to, we call a *structural footnote* a text extension attached to a portion of a base text. In modeling texts with complex semantics, it is often preferable to characterize precisely the portion of text a *footnote* refers to. This increases the accuracy of the *footnote*. Structural footnotes can be supported by view-aware XML browsers in various manners: For example, the request to display the footnote – as a stretch or pop-up text – might be accompanied by a highlighting of its base text. In addition, views as defined above, provide with a framework for a semantic typing of *footnotes*. Both notions, structural footnotes and semantic typing of footnotes contribute to a better modeling of complex technical documents.

3. SEMANTICS VS. NAVIGATION

3.1 Link Model vs. Browsing Model

Today's hyperlink models (such as HyTime [9] or the hyperlink model subjacent to HTML) can be separated into a (semantic) link model, for which links are semantic relationships, and a browsing model, devoted to specifying a hypertext traversal behavior of links. This distinction is similar to that between content and rendering, expressed using, respectively, a generic markup language such as XML and a style sheet language such as CSS, DSSSL or XSLT/XSL-FO. Distinguishing between a link model and a browsing model increases both data and device independence.

Consider the following example of an XLink [10] link:

```
<link xlink:type="extended">
 <locator xlink:type="locator"
  xlink:href="rep.xml#xpointer(abstract)"/>
 <locator xlink:type="locator"
  xlink:href="tec-doc.xml"/>
 <arc xlink:type="arc"
  xlink:show="new"/>
</link>
```

This link defines both,

- a semantic relationship between (a set) of data items (namely, an XML element and a whole XML document) – this refers to the *link model*;
- the behavior of the hyperlink on traversal (opening a new window for the target) – this refers to the *browsing model*.

Using XLink, aspects of both models are specified together. Links can be labelled semantically using the *semantic attributes* `role`, `arcrole` and `title`. Aspects of the browsing model are specified through the *behavior attributes* `show` and `actuate`.

If aspects of both models are specified separately, it is possible to bind different browsing behaviors to the same link. This is useful since the same document is likely to be browsed not only using standard-sized computer screens, but also the tiny screens of cellular phones and hand-held

computers as well as the large projection screens for which document browsing is likely to take new forms. Thus, the distinction between link and browsing models increases device independence. It also increases the independence between data modeling and data usage. Data and device independence are key issues in data and document modeling.

The distinction between link and browsing models is analogous to that between content and rendering in XML: The logical structure of a document is expressed in XML, while the rendering is expressed using style sheets. Just as standard style sheets specify the rendering of documents, there ought to be a *browsing style sheet* defining the hyperlink behavior. In addition, a link language is needed.

To stress both aspects of links, semantic and behavior, *hyperlink* refers below to traditional hypertext links while *link* refers to semantic relationships (that possibly might be assigned a hypertext behavior).

3.2 Link Language

It is fairly simple to obtain a link language: It suffices to consider any hyperlink specification language such as XLink without its behavior features (i.e., without the attributes `show` and `actuate` in case of XLink). This means basically, that there is no need to change existing link languages, one just has to leave out a very small part of it. With such a simplified link language, it is no longer possible to define any hyperlink behavior. Using such a "Simplified XLink", links can only be characterized semantically through the semantic attributes `role`, `arcrole` and `title`. In standard XLink, the attribute `show` specifies behavior. In Simplified XLink, the purpose of this attribute could be achieved by a semantic attribute as follows (instead of `show`, a more appropriate name could be `extension`):

- `new`: link to additional, out-of-context information
- `embed`: link to additional, in-context information
- `replace`: link to alternative descriptions of the same information

In this case, the distinction between out-of-context and in-context information refers to semantic contexts: For example, a link from text T_1 explaining an industrial maintenance procedure to some out-of-context (in-context, resp.) text T_2 means that the additional information T_2 can (cannot, resp.) be understood without the information provided by the origin T_1 of the link.

3.3 Browsing Style Sheets

A browsing style sheet for a document can for example assign to the three values of the `show` (or `extension`) attribute the same behaviors as in standard XLink. Another browsing style sheet for the same document can assign other behaviors.

Expressed in Simplified XLink, the example given above becomes:

```
<link xlink:type="extended">
 <locator xlink:type="locator"
  xlink:href="rep.xml#xpointer(abstract)"/>
 <locator xlink:type="locator"
  xlink:href="tec-doc.xml"/>
 <arc xlink:type="arc"
  xlink:title="new information"/>
</link>
```

This link still defines the same semantic relationship as above between a certain element of an XML document and another XML document. However, it no longer specifies that a new window has to be opened on traversal of that hyperlink. It simply (semantically) labels the link as *new information*.

A browsing style sheet language has been defined as a simple extension of Cascading Style Sheets (CSS), called *BCSS*. Indeed, CSS is a simple though powerful mechanism for adding style (for example fonts, colors, spacing) to XML documents (amongst others). In CSS, this is achieved by assigning property/value pairs to document elements. *Browsing CSS* (BCSS) [4], allows to assign browsing features, i.e. traversal behavior, to (semantic) links.

In this browsing style sheet language, the same hyperlink behavior as that of the first example given above can be specified as follows:

```
[xlink:title='new information'] { show: new }
```

This means that only two new properties (namely `show` and `actuate`) and the according values have been added to CSS. Therefore, browsing style sheets can be added easily to existing standards like HTML, XML, CSS and XSLT and also to software systems like web browsers and servers.

Other approaches for defining browsing style sheets, such as one based on XSLT/XSL-FO, have also been considered [4]. XSLT/XSL-FO does not even require any extension.

A browsing style sheet is associated to an XML document (with links expressed in Simplified XLink) by a processing instruction in the very same way as with other style sheets.

Note that it is intended to associate browsing style sheets with an XML document, not with a linkbase. Specifying (semantic) links, a linkbase does not have to express any browsing behaviors. Thus, different documents can use the same linkbase with different browsing style sheets. This makes it possible for distinct documents to rely upon the same linkbase, yet having different hyperlink behavior. This contributes to device independence, for browsing is likely to take different forms on different devices.

4. READER'S VIEWS

4.1 Restructuring While Reading

Reader's View is a tool with which a reader can restructure a document (within certain limitations given by the document's author or the browser). Reader's View is somewhat similar to an XML editor included within a browser, but with one difference: Reader's View leaves the original source document unchanged. For example, when the reader decides to leave out a certain paragraph of text (cf. below), this paragraph will not be erased from the source document, but Reader's View stores the information not to display this paragraph anymore.

Despite of their names, reader's views (which are built using Reader's View) are fundamentally different from author's views (cf. Section 2). Author's views are created by the *author* of the document at writing time. A reader's view, in contrast, is created by the *reader* of the document while browsing.

4.2 Features

Using Reader's View, the nodes of the source document tree may be selected or rearranged. For example, a certain

node can be left out or put in front of another node. Restrictions to such operations can be expressed. For example, one can forbid restructuring violating parent-child relationships or the selection of parts (rather than the whole) of nodes.

Explicit construction of a reader's view allows the reader to *pick up* certain nodes from the source document and add them in any order to the reader's view under construction. The nodes are not copied from the source document to the reader's view, but referenced via links.

Thus, Reader's View can be seen as the generalization of today's major browsers' bookmark functionality. Traditional bookmark tools make it possible to store a set of links. With Reader's View one can in addition store arbitrary nodes, in particular text nodes written by the reader, as well. Such nodes can be linked to nodes of the document(s) from which the reader's view was constructed, thus yielding a powerful annotation mechanism.

Moreover, the nodes added by a reader can contain their own links. These links may point to nodes contained in the original source documents or to nodes that are only contained in the reader's view under construction. Thus, using Reader's View a reader can build up his own structured document containing new text and links as well as text, links and structure from the original source document. Thus Reader's View can be seen as a kind of *electronic notebook*.

4.3 Implementation with XLink

A reader's view is simply an XML document. This means that it can be viewed with any browser, even if the browser is not reader's view-aware. Specific to reader's views is how they are created. This is a major difference to ordinary annotation systems like [1], which rely on special annotation servers.

If a reader selects a certain node from the original document, an outbound link with the selected node as its ending resource is created. This link is inserted into the reader's view. If the user wants a certain node not to be included in the reader's view, no link is created for that node.

The behavior attributes of XLink or of a browsing style sheet (cf. Section 3) can be used to implement the features of a reader's view. They allow a reader to see a reader's view document as if its nodes were copied from the original document, even though they are only referenced by links.

Nodes created by the reader, i.e. "annotations", cannot be referred to in a reader's view document just as links, as the nodes' content has to be stored as well. The content of such nodes is copied into the reader's view document as it is entered by the user.

5. PROTOTYPE

A prototype called *DT Browser* has been implemented in Java. *DT Browser* is a prototype for an XML Browser with support for browsing style sheets and Reader's View as described in this article. *DT Browser* and the principles underlying it are described in detail in [4].

A prototype for author's views is currently under development. It makes use of CSS style sheets to select between different views. This allows this prototype to be used in standard web browsers.

6. ACKNOWLEDGMENTS

The authors thank Kazuhiro Kitagawa and Martin Dürst, both Keio University and W3C, for useful suggestions and also Norbert Eisinger, University of Munich, for his help in developing the prototypes.

7. REFERENCES

[1] Annotea Project. http://www.w3.org/2001/Annotea/.
[2] P. Brusilovsky. Methods and Techniques of Adaptive Hypermedia. In *User Modeling and User-Adapted Interaction*, volume 6, 1996.
[3] J. Cowan et al., eds. XML Information Set. W3C Recommendation, October 2001. http://www.w3.org/TR/xml-infoset/.
[4] M. Kraus. A Toolkit for Advanced XML Browsing Functionalities. Master's thesis, University of Munich, 2000. http://www.pms.informatik.uni-muenchen.de/publikationen/#DA:Michael.Kraus.
[5] M. Sawicki et al., eds. Ruby Annotation. W3C Recommendation, May 2001. http://www.w3.org/TR/ruby/.
[6] P. De Bra et al. An Extensible Data Model for Hyperdocuments. In *4th ACM Conference on Hypertext*, 1992.
[7] P. De Bra et al. Aham: A Dexter-based Reference Model for Adaptive Hypermedia, 1999.
[8] F. Halasz et al. The Dexter Hypertext Reference Model. *Communications of the ACM*, 37(2):30–39, 1994.
[9] S. R. Newcomb et al. The "HyTime" Hypermedia/Time-based Document Structuring Language. In *Communications of the ACM*, volume 34, November 1991.
[10] S. DeRose et al., eds. XML Linking Language (XLink). W3C Recommendation, June 2001. http://www.w3.org/TR/xlink/.

Biography

François Bry, born 1956, is currently investigating database methods and applications emphasizing semistructured data, document modeling, and query answering. Formerly, he worked on deductive databases, logic programming, and automated theorem proving. Since 1994, he is a full professor at the Institute for Computer Science of the University of Munich, Germany. Before joining the University of Munich, he has been working with the European Computer-Industry Research Centre (ECRC), Munich, and a few other companies in Paris, France. In 1981 he received a PhD from the University of Paris. He has been visiting at several university and research centers, including ICOT in Tokyo. He contributed to scientific conferences as an author, program committee member or chairman. He enjoys biking, hiking, and traveling.

Michael Kraus, born 1975, is currently working on his PhD in the field of web application modeling at the Institute for Computer Science of the University of Munich, Germany. In his diploma thesis, he developed a toolkit for advanced XML browsing functionalities. Before, he visited Nagoya University, Japan, where he worked on brain computer interfaces.