Assembling Documents from Digital Libraries*,**

Helena Ahonen, Barbara Heikkinen, Oskari Heinonen, and Pekka Kilpeläinen

University of Helsinki, Department of Computer Science
P. O. Box 26, FIN–00014 University of Helsinki, Finland

Abstract. We consider assembling documents using, as a source, a digital library containing SGML documents. The assembly process contains two parts: 1) finding interesting fragments, and 2) constructing a coherent document. We present a general document assembly framework. First, we describe a system for tailoring control engineering textbooks. Its assembling facilities are rather restricted but, on the other hand, the quality of documents produced is high. Second, we address the problem of filtering and combining interesting information from a large heterogeneous document collection. The methods presented offer various ways to find the interesting document fragments. Moreover, the elements found in the fragments are mapped to generic elements, like sections, paragraph containers, paragraphs and strings, which have known semantics. Hence, even arbitrary compositions can be formatted and printed.

1 Introduction

In an on-going research and development project called Structured and Intelligent Documents (SID) [1] we study *document assembly* in its different aspects. By document assembly we mean computer-aided construction of a new document using several existing document sources. Assembly is usually an interactive process (Figure 1) within which an author or editor uses various tools to find appropriate sources and to configure the intended document. The process consists of two parts: 1) finding the interesting document fragments and 2) constructing a coherent document from these fragments.

One application area we have considered is assembling educational material from digital libraries. A digital library may contain, for instance, textbooks, articles, exercise collections, and simulations. This kind of a library can be used in multiple ways. The publisher can print customized textbooks for various school branches and levels. Additionally, teachers can form tailored compositions of WWW or CD-ROM delivered material—e.g., summaries, slides and tests—for their own classes. Students could also use the digital library directly for problem-oriented learning, as a data bank, or as an active textbook with computer-assisted learning features.

Recently, several publishers and universities have launched their own services for delivering customized textbooks [10,11,3]. All the approaches seem to have

* This work was supported by the Finnish Technology Development Centre (TEKES).
Authors' e-mail: *{hahonen,bheikkin,oheinone,kilpelai}@cs.helsinki.fi*.

** To appear in *Proceedings of the 8th International Conference and Workshop on Database and Expert Systems Applications (DEXA 97)*, Toulouse, France, September 1997.

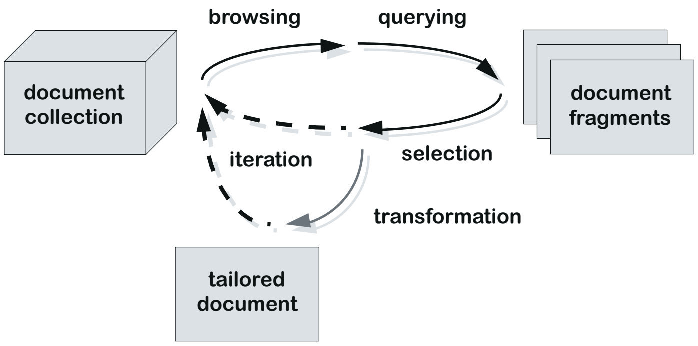

Fig. 1. Assembly process.

a fairly static set of document fragments that can be combined to form a book, and their user interfaces simply allow the user to choose a selection of these parts. We claim that dynamic compositions are needed, and to achieve this, we have decided to use Standard Generalized Markup Language (SGML) [6] for the management of the material. Haake et al. have introduced the Individualized Electronic Newspaper [5] which also uses SGML as the document presentation format. Their approach, although somewhat similar to ours, concentrates on gathering and presenting news-like material according to readers' profiles while our goal is more general: a coherent assembled customized SGML document.

SGML is a metalanguage for defining document structures. The logical structural parts, *elements*, of an SGML document are marked up by start and end *tags*. The set of element names and the permitted structures of the elements are given in the *document type definition* (DTD) that is essentially a context-free grammar, in which the right-hand sides of the rules are regular expressions. A DTD can be used to facilitate structured queries and various transformations needed, e.g., to produce multiple output formats. SGML representation defines only the syntax of the structures: any semantics, e.g., how the elements should be formatted for printing, has to be attached by some application.

Although one of the strengths of SGML technology is to allow several representations and formats to be generated from the same documents, multiuse of documents by dynamic assembly of structure elements is still an unsolved problem, and hardly any techniques or tools exist. The necessity of assembly is, however, clearly seen and considered to be of great importance [8,9].

In this paper we present a general document assembly framework. First, we describe a system for tailoring control engineering textbooks. Its assembling facilities are rather restricted but, on the other hand, the quality of documents produced is high. Second, we address the problem of filtering and combining interesting information from a large heterogeneous document collection. The method presented offers various ways to find the interesting document fragments. Moreover, the elements found in the fragments are mapped to generic elements, like sections, paragraph containers, paragraphs and strings, which have known semantics. Hence, even arbitrary compositions can be formatted and printed.

The rest of the paper is organized in the following way. Section 2 describes the system for tailoring textbooks via a WWW user interface. The general framework is introduced in Section 3. Finally, Section 4 gives some conclusions.

2 System for Tailoring Textbooks via WWW

One of our project partners, the Finnish National Board of Education, has started to create a new digital library that is intended to contain various educational material related to control engineering. We have participated in converting textbooks into SGML, and we have also implemented a system for tailoring new specialized textbooks using the material. The following section presents the structure of the documents and how it can be used in assembling. Section 2.2 describes the system architecture.

2.1 Structure of the Documents

The first components of our digital library are three textbooks on control engineering. The books, originally prepared using MS Word[1], contain many equations and technical pictures. Creating a structured SGML version required laborious conversion steps using tools such as MathType[2], DynaTag[3], OmniMark[4], and FrameMaker+SGML[5].

The major hierarchical structure elements of a book are chapters, sections, subsections and paragraphs. We have used as a basis for our document type definition the Book DTD of the ISO 12083 standard [7] extended with element types for exercises, answers, examples, formulas, definitions and clauses. The book is enriched with internal supplementary information. Manual seeding, based on information from the authors, was necessary to classify the fragments of the material into appropriate categories. Each text paragraph is classified according to the level (vocational, college, university), the branch of specialization (metal industry, process industry, data communications, civil engineering, etc.) and the didactic form of the contents (introductory, base material, advanced, applied, summary, auxiliary, examination).

The categorization is represented in the text paragraphs and their superstructures as SGML attributes. Attribute values are propagated from the upper levels of the structure, if not overridden in the element. Figure 2 is an example of the supplementary information attached to the markup of the material.

The shown chapter was authored by Jari Savolainen (author=aJS). It is recommended for students at the college level (level=b). The content of the

[1] MS Word is a trademark of Microsoft Corporation.
[2] MathType is a trademark of Design Science, Inc.
[3] DynaTag is a trademark of Inso Corporation.
[4] OmniMark is a trademark of OmniMark Technologies Corporation.
[5] FrameMaker+SGML is a trademark of Adobe Systems Incorporated.

```
<CHAPTER id="c4" author="aJS" level="b">
  <TITLE>S&Auml;&Auml;T&Ouml;LOHKOT TAAJUUSTASOSSA</TITLE> ...
  <SECTION id="c4.1" level="b" content="BB" branch="unspec">
    <TITLE>Impedanssi-kompleksinen vastus</TITLE> ...
    <EXAMPLE id="c4ex1" level="c" content="CC" branch="unspec">
      <FIGGRP id="c4fg2"><FIG entity="c4f2"><CAPTION>Impedanssin
        sis&auml;lt&auml;m&auml; ...</CAPTION></FIGGRP>
      <P><KEYWORD freq=1>kokonaisimpedanssi</KEYWORD>
        Laske sarjakytkenn&auml;n kokonaisimpedanssi U = 5 V</P>
      <ANSWER><P><KEYWORD freq=1>kokonaisimpedanssi</KEYWORD>
        Kokonaisimpedanssi on sarjaankytkettyjen impedanssien summa.
        <EQ id="c4df13"><EQBODY entity="c4d13"></EQ></P> ...
```

Fig. 2. An example of internal supplementary information.

section 4.1 is base material (content=BB). The example of the section is directed at students at the university level (level=c) and it contains advanced material (content=CC). Keywords and their frequencies are listed explicitly. Formulas are in external entity files (attribute *entity* of element *eqbody*) in MIF format. They can also be represented in picture formats, e.g. TIFF. Using the added markup, we can easily extract, e.g., base material for college level students, or exercises for university level students. Search features of existing commercial SGML browsers such as DynaText[6] or Panorama Pro[7] support this kind of querying.

2.2 System Architecture

We have built a system that gives teachers a possibility to order customized control engineering textbooks. An overview of the system can be seen in Figure 3. An order is processed as follows.

1. **Order.** The client fills in an HTML form and submits it.
2. **Processing of order.** The order form is processed. The composition of the assembled document is created based on the input from the client and some heuristic rules. The table of contents is returned to the client. If the client confirms the order, the specification of the composition is submitted to the assembly process.
3. **Assembly.** According to the input for the assembly, which is represented by a list of file names, the assembly process retrieves the desired texts and images from the SGML collection and composes the new textbook. Thereafter, the textbook is formatted and the page layout is manually checked. Finally, a PostScript file is submitted to the printing house.
4. **Printing.** The printing house produces the textbooks and delivers them to the client.

[6] DynaText is a trademark of Inso Corporation.
[7] Panorama Pro is a trademark of SoftQuad, Inc.

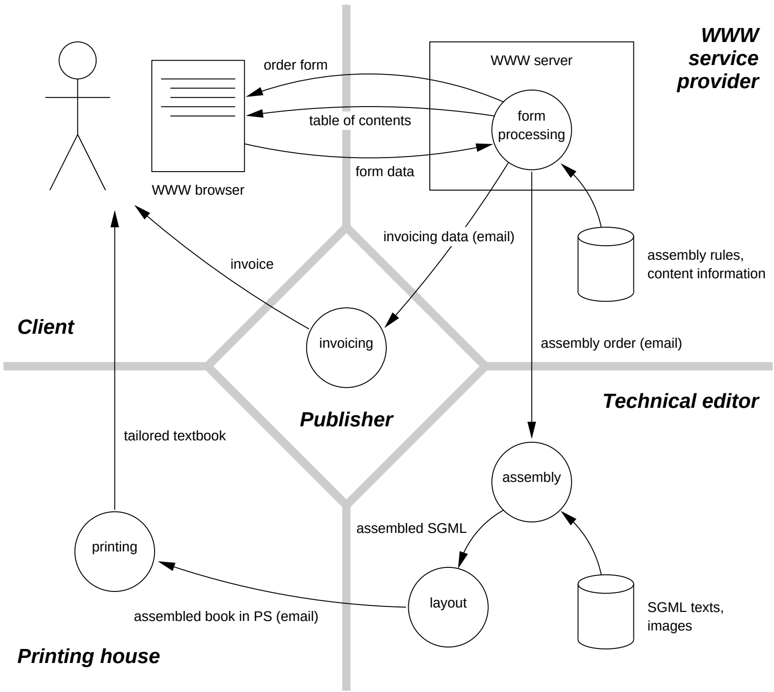

Fig. 3. Overview of the system.

As a user interface for the teachers we use an HTML form (Figure 4). The form contains a table of contents for each of the three books. The fragments that can be used for selection are chapters, sections, subsections, and, in some cases, author-defined sequences consisting of several paragraphs.

The columns in the table of contents are (from left to right):

1. **Choice.** By checking choice buttons the user selects corresponding structures of the books. The user can select whole chapters, or just some sections, subsections, or predefined paragraph sequences.
2. **Title or beginning of text.** Original chapter, section or subsection title, or the beginning of the text of the paragraph sequence.
3. **Page count.** Length of the part.
4. **Content.** For example, 'johdanto' (introductory), 'perusaines' (base material), 'soveltava' (applied).
5. **Level.** For example, 'ammattioppilaitos' (vocational).

The rightmost column indicates the lowest school level for which the part is recommended. At the moment this knowledge is only given to the teacher as a hint; it is not used in any automated way. As the digital library does not contain any branch-specific material yet, we have not utilized the values of the *branch* attribute.

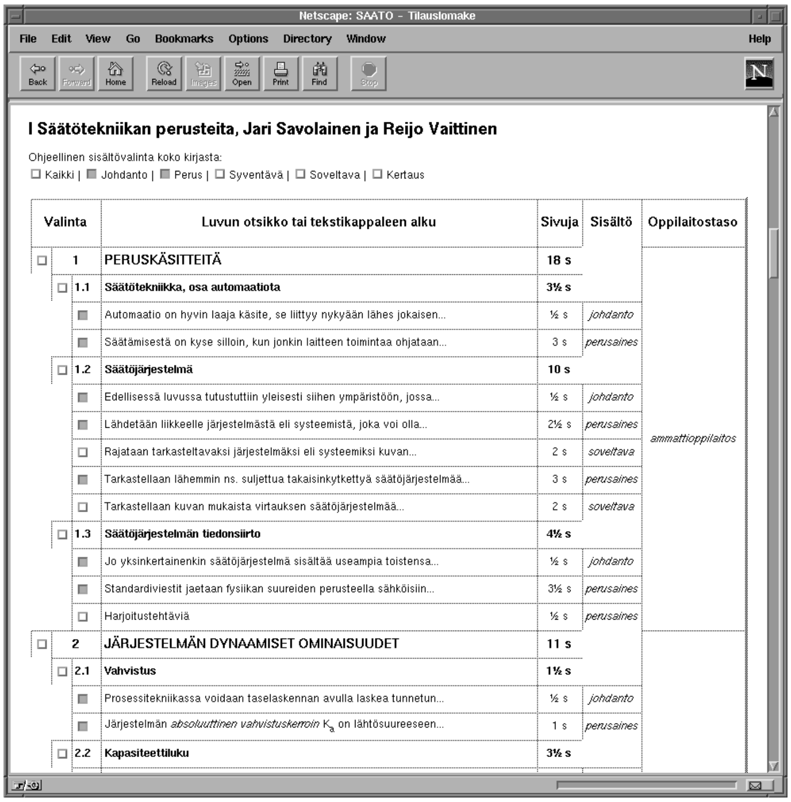

Fig. 4. Order form.

The six choice buttons above the table are shortcuts. They enable the user to select multiple items by one click. The first button selects all the material, and the others select all parts of the named content (e.g., 'johdanto'), respectively. This selection approach could be extended to other attributes as well.

As the selections have several dimensions, it is not at all obvious, how all the choices should be presented to the client; the order form becomes easily too complicated. One solution would be to offer pre-defined assemblies as a starting point: some fragments would be pre-selected but the user could change the selections.

In order to achieve sensible results, the assembly has to utilize heuristics which guarantee the inclusion of all necessary fragments. For example, if only some sections of a chapter are selected then also the title of the chapter and the introductory paragraphs are included.

If the ordered books are delivered on paper, the problem of predictability arises (since attributes and heuristics are used). Hence, the teacher should be able to check the contents of the result. At the moment, after the teacher has submitted the form, the ordering system immediately returns the table of contents for the assembled textbook, as well as the number of pages and the price of the order. We also assume that the teacher has a paper copy of all the original books, when he/she selects the fragments to be included. Instead of the table of contents, the whole assembled textbook could be sent to the client, e.g., by email.

After the confirmation of the order, a list of file names is submitted to the assembly process. The assembly retrieves the required files and concatenates them to form one SGML document. As the DTD of all the assembled textbooks is the same as the original, it is possible to create, using FrameMaker+SGML, the formatting layout nearly automatically. Due to some problematic cases, e.g., improper page breaks, a manual inspection is still needed. Finally, the textbooks are produced by a printing house. In our system a preprinted cover page is used, on which the personalized headings are printed. The printing house delivers the books to the client.

3 General Assembly Framework

A tailoring system presented in the previous section produces high quality textbooks, but its applicability is restricted to fairly small document collections. Whereas, when large and heterogeneous digital libraries are concerned, the user should be offered more flexible ways to filter information and construct meaningful combinations.

3.1 Selection of Interesting Document Fragments

As we have seen, in the current system the user selects chapters and sections by marking them up in the table of contents. This is not feasible if the document collection is large. Hence, our new system under development allows four ways to select elements to be included in the assembly.

The first one is a rather standard search that will include full-text search with conditions on the structure and attribute values. For instance, within our control engineering library, the user might want to create a collection of exercises for college students majoring in data communications.

The second way is based on the Scatter/Gather clustering [4]. The user-controlled iterative clustering process forms a classification of the material, and gives the user an extracted view of the documents. This is especially invaluable when the user is dealing with an unfamiliar collection of documents; traditional query models do not help the user to achieve a simple overview. The user can further select the appropriate clusters for reclustering, and in this way narrow down the retrieval space.

The third possibility is to browse the document collection by navigating in the tree structure and selecting the desired elements manually. Reasonable use of this selection usually necessitates that the two above-mentioned selection methods have already been used to reduce the document collection.

The fourth possibility is to start from the elements selected so far, and let the system search for similar or related elements in the collection. Similarity of two elements can be estimated, for example, by the amount of the same words they contain.

In the current system the user can only prune the existing books: mixing the sections of different books is not possible, and even within one book it is not possible to change the order of sections. This kind of modifications are, however, often desired. Hence, in the new system the user is allowed to modify the set of selected documents in order to obtain a final textbook. The modifications include rearranging the elements, moving or copying selected fragments to be part of an existing element, and replacing an element either by its parent or children elements. As the result of the selection phase, an ordered list of document fragments, with the original internal tree structures, is returned.

3.2 Constructing a Coherent Document

In our textbook tailoring system, all the documents share a common DTD, and also the resulting textbook is of the same type. Thus, the overall structure of the new document is known, and we can also utilize the formatting declarations of the original documents. This is not the case in general. The source documents may have differing structures, and an arbitrary composition of their fragments does not belong to any known document class. Therefore, no formatting rules are available, i.e., the documents cannot be browsed or printed in the formatted form.

When constructing a useful SGML document, two points have to be considered: 1) what is the document type definition of the new document, and 2) how the application programs, e.g. formatters, can process the document. Given a set of document fragments, each of which is a valid SGML element according to its DTD, the following steps are taken.

1. The elements of the fragments are classified and generalized.
2. A new document type definition is constructed.
3. A new document is constructed using the generic elements.
4. Formatting rules for the assembled document are constructed.

The aim of the element classification is to map every element to some generic element with well-known semantics. We have defined a set of generic elements that often appear in texts. We also give them element declarations and simple semantics, e.g., how the element should be printed.

All the fragments are traversed once to collect the set of element names, and for each element, the set of elements it contains as well as the average length of its content. Thereafter, each element found is mapped to some generic element: *Section*, *Paragraph container*, *Paragraph*, or *String*. Mapping is done bottom-up using the rules in Table 1. First, all short elements are classified as *Strings*. Second, *Paragraphs* are identified. After that, all the remaining elements are classified as *Sections*, and finally, all *Sections* that do not contain any other *Sections* are remapped to be *Paragraph containers*.

Generic element	Conditions
String	Contains ordinary text and/or other *Strings*; length of the content is less than 100 characters.
Paragraph	Contains ordinary text and/or *Strings*; length of the content is more than 100 characters.
Paragraph container	Contains *Strings* and/or *Paragraphs* (at least one *Paragraph*).
Section	Contains *Strings, Paragraphs, Paragraph containers* and/or *Sections* (at least one *Paragraph* or *Paragraph container*).

Table 1. Recognition rules for element classification.

Some elements, like tables and figures, do not match well any of the generic elements. These elements are preserved as such, as well as the elements they contain. The elements can be identified using simple heuristics, e.g., in tables the proportion of tags compared to the text content itself is high, and the DTD usually gives enough hints for recognizing figures and formulas.

Construction of the new DTD is straightforward: it expresses essentially nothing more than the inclusion relationships of the generic elements, e.g., a section is allowed to contain paragraphs but not vice versa (see Figure 5). *Text* refers to SGML character data.

```
Document -> (Section | Paragraph_container | Paragraph | String)*
Section -> (Section | Paragraph_container | Paragraph | String)*
Paragraph_container -> (Paragraph | String)*
Paragraph -> (String | Text)*
String -> (String | Text)*
```

Fig. 5. Grammar of the new DTD.

If some element structure is left unclassified, its original definition is added to the DTD. Additionally, the element has to be contained in the definition of the parent. For instance, if a *Table* has occurred within a *Paragraph container*, the definition for *Paragraph container* is changed to:

```
Paragraph_container -> (Paragraph | String | Table)*
```

The actual assembly process composes a valid SGML document from the fragments. After classification we have an ordered list of fragments containing classified elements. The root elements of the fragments may be *Sections, Paragraph containers* or *Paragraphs.* (*Paragraph* is the smallest fragment size allowed.)

First, a new document root is constructed for the new document. Our intended top-level structure contains a list of *Sections*, possibly preceded by a couple of *Paragraphs* or *Paragraph containers.* Hence, *Paragraphs* and *Paragraph containers* located between two *Sections* in the fragment list have to be combined by generating a new *Section*. If the generated new *Section* is too large, the *Paragraphs* may be joined in order to get several smaller *Sections* [2]. Additionally, the desired number of top-level *Sections* can be given as a parameter. If the number of *Sections* and generated *Sections* exceeds this threshold, some consecutive *Sections* have to combined. The *Sections* to be combined can be chosen by considering the similarity of consecutive *Sections*, based, e.g., on the amount of words in common.

The formatting rules can be constructed in a straightforward manner. The generic elements have their formatting rules that are independent of the documents assembled, whereas an unclassified element is output using the rules attached to the original document. The formatting of generic elements is based on the following principles. *Strings* are inline elements (no newline at the end). The font of ordinary text is normal, whereas marked *Strings* are emphasized using, e.g., boldface or italics. *Paragraphs* have a newline at the end. *Sections* have some vertical space at the beginning, and if a *Section* has a *String* element in the beginning, it is interpreted as a title and hence output using boldface and large font. *Sections* are numbered automatically.

The mapping of elements into generic elements, which have simple formatting rules, can be compared to the conversion of some structured form to an HTML format: the complexity of the structure is reduced. Opposite to HTML, the new document still has a hierarchical structure. Moreover, the old element name can be stored to each element as an attribute name. Hence, the original structure can be reconstructed.

4 Conclusions

We have introduced a novel approach for assembling documents using, as a source, a digital library containing SGML documents. The assembly process contains two parts: 1) finding interesting fragments, and 2) constructing a coherent document.

We first presented a system for tailoring textbooks via WWW. In our system, the order of a client is processed as follows. The client fills in and submits an HTML form. The form is processed and the fragments to be included in the assembled book are selected automatically. The choice of fragments is dependent on the client's input and a few heuristic rules. The texts and images are retrieved from the digital library and the layout is created. Finally, the books are printed and delivered to the client. We have developed our assembly system together with a group of teachers, including the authors of the textbooks used. At the moment, the teachers are about to select sample assemblies of material. So far the results are encouraging: the teachers find the assemblies useful, even if the collection is still rather restricted.

The approach demonstrated with the current system will be developed further as part of a more general assembly environment. Since document collections concerned are assumed to be large, more advanced selection possibilities are included in the system. Additionally, flexible assemblies from several sources have to be supported. Our method has the following steps.

1. The user selects interesting document fragments by querying and browsing.
2. The elements of each fragment are mapped to well-known generic elements (section, paragraph container, paragraph, string).
3. A new document type definition is formed.
4. A new document is constructed from the generic elements.
5. The formatting rules are formed for the elements.

The approach presented is simple and easily implemented. Each of the above steps can be further developed to gather more of the semantics of the source documents. For instance, new generic elements, e.g. tables, can be recognized. Our aim is to improve the intelligence of the classification system, the DTD generation, the assembly heuristics and the formatting rules. However, as the primary goal is to present a compilation of information in an organized form with uniform and clear output, a rather simple method may actually work, at least in unanticipated situations, more reliably.

References

1. Helena Ahonen, Barbara Heikkinen, Oskari Heinonen, Jani Jaakkola, Pekka Kilpeläinen, Greger Lindén, and Heikki Mannila. Intelligent Assembly of Structured Documents. Report C-1996-40, Department of Computer Science, University of Helsinki, 1996.
2. Helena Ahonen, Barbara Heikkinen, Oskari Heinonen, and Mika Klemettinen. Improving the accessibility of SGML documents: A content-analytical approach. In *SGML Europe '97*, Barcelona, 1997. GCA.
3. Custom CourseWare. McMaster University Bookstore, 1997. URL: `http://bookstore.services.mcmaster.ca/home/ccw/ccw.html`.
4. Douglas R. Cutting, Jan O. Pedersen, David Karger, and John W. Tukey. Scatter/Gather: A cluster-based approach to browsing large document collections. In *Proc. of the 15th ACM/SIGIR Conference*, Copenhagen, 1992.
5. Anja Haake, Christoph Hüser, and Klaus Reichenberger. The individualized electronic newspaper: an example of an active publication. *Electronic Publishing — Origination, Dissemination and Design*, 7(2):89–111, June 1994.
6. ISO. *Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML), ISO 8879*, 1986.
7. ISO. *Information and documentation — Electronic manuscript preparation and markup, ISO 12083*, 1994.
8. W. Eliot Kimber. Re-usable SGML: Why I demand SUBDOC. In *SGML '96*, Boston, 1996. GCA.
9. John McFadden. Hybrid distributed database (HDDB) and the future of SGML. In *SGML Europe '96*, Munich, 1996. GCA.
10. Nelson Canada Power Pak. Nelson Canada, a Division of Thomson International, 1997. URL: `http://www.thomson.com/nelson/custom/custom.html`.
11. Primis. Primis Custom Publishing, a Division of McGraw-Hill, 1997. URL: `http://www.mhcollege.com/primis/`.