Design and Implementation of a Document Assembly Workbench*

Helena Ahonen[1], Barbara Heikkinen[2], Oskari Heinonen[2], Jani Jaakkola[2], Pekka Kilpeläinen[2], and Greger Lindén[2] **

[1] University of Tübingen, Wilhelm-Schickard-Institut für Informatik, Sand 13, D–72076 Tübingen, Germany
[2] University of Helsinki, Department of Computer Science, P.O. Box 26 (Teollisuuskatu 23), FIN–00014 University of Helsinki, Finland

Abstract. Computers support the management of large collections of text documents, but efficient reuse of document collections for producing new documents remains inherently difficult. We describe and discuss the design and implementation of a document assembly system based on a document assembly model, where the user produces new specialized documents by querying and browsing a collection of structured document fragments.

1 Introduction

The role and the concept of documents have undergone a tremendous change. A document of today can be an active and dynamic entity, which is not only created by a user, but also collected, combined and customized from reusable documents [14]. Progress in data technology has greatly facilitated creating, storing, retrieving and disseminating individual documents, but the problem of easily and effectively reusing existing and often large document collections remains still largely unsolved.

One of the basic problems is to provide, or at least to support the on-demand generation of individualized documents through dynamic document assembly. By *document assembly* we mean computer-aided construction of new documents from a collection of document material. Such reuse includes finding the relevant document building blocks, modifying them as needed, and stitching the pieces together to make a new document. The lack of precise computational and conceptual models of document assembly hinders the development of widely applicable systems. Even though many corporate-level document assembly activities are under way (see, e.g., [17]), no generally accepted models of document assembly exist yet.

* This work is supported by the Finnish Technology Development Centre (TEKES) and seven Finnish enterprises (Aamulehti Group, Oy Edita Ab, National Board of Education, WSOY, Helsinki Media, Lingsoft, Inc., and MTV3). Moreover, the grants by the Academy of Finland (Helena Ahonen), the Finnish Cultural Foundation (Barbara Heikkinen), and the 350th Anniversary Foundation of the University of Helsinki (Oskari Heinonen) have provided additional support.

** Authors' e-mail: *{Helena.Ahonen, Barbara.Heikkinen, Oskari.Heinonen, Jani.Jaakkola, Pekka.Kilpelainen, Greger.Linden}@cs.Helsinki.FI*.

We describe a document assembly model and architecture, which we have developed to study principles and methods of intelligent document reuse. The work is done in an on-going research and development project called "Structured and Intelligent Documents (SID)". Document assembly is a central goal application of the project. As the basis for the project we consider structured documents marked up using SGML.

Standard Generalized Markup Language (SGML) [10] is an international standard [11] for defining application and platform independent document markup languages, and for encoding documents using the defined markup. An SGML application is based on a Document Type Definition (DTD), which defines the structure of a class of related documents and the markup used for indicating them, and upon which the manipulation of the documents can be based. SGML has been widely adopted by industry since it offers a framework for creating structured documents with high potential for longevity and multi-purpose reuse. The SGML standard does not, however, provide ready solutions for managing or processing of large document collections. Although one of the strengths of the SGML technology is to allow several representations and formats to be generated from the same documents, reuse 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 [13, 15].

We have developed a document assembly framework based on versatile recognizing and manipulating of *document fragments*, which are consistent and relatively independent document parts used as the basis for new assemblies. Similar ideas recur at document manipulation and IR meetings under different names, such as passages, semantic fragments, information units, minimal revisable units, or micro documents. The assembly framework is not inherently SGML specific, but the possibility of using generic semantic document markup and the existence of tools for a standardized formalism makes the implementation of a document assembly system based on the fragment framework much easier. Our approach to document assembly is closely related to the work reported by Davis and Hey [8]. They consider the extraction of a set of multimedia documents from a digital library and their presentation as a single hypermedia document. Our work differs from theirs, e.g., in the employment of generic SGML structures to allow the assembly result to be further transformed to the final delivery format, be it digital or hard-copy.

The rest of this paper is organized as follows. In the next section we present our document assembly model and discuss additional information that can be used to support intelligent assembly. Then in Sect. 3 we describe our document assembly prototype, its user interface, architecture and implementation. Finally, we give a short conclusion and discuss some open problems.

2 Intelligent Document Assembly

Document assembly means computer-aided construction of new documents using several existing document sources. Assembly is usually an interactive process (Fig. 1), where an author or editor uses various tools to find appropriate sources and to configure the intended document. The process consists of selecting, combining, and modi-

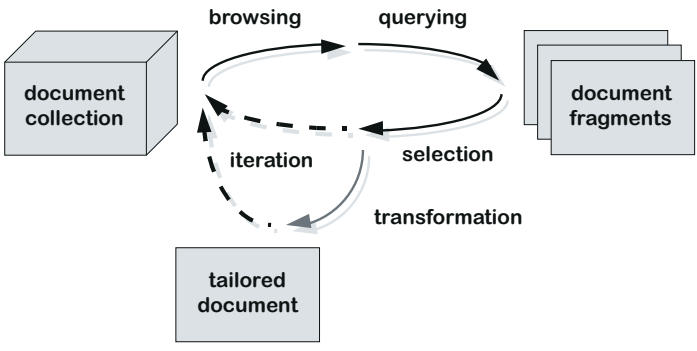

Fig. 1. Document assembly process.

fying document components. The document composer works iteratively on a document collection until a result document is obtained. Even though producing carefully edited publications requires human creativity, we believe that increasingly challenging assembly tasks should, and could, be supported much more by computerized methods than that are available today.

The assembly process is preceded by initialization actions to populate the document collection and to analyze and preprocess the documents for the assembly. We allow heterogeneous document collections containing documents with differing document types. Moreover, the set of DTDs used is not fixed, e.g., no DTD specific semantics is assumed to be available. Hence, the assembly process has to rely on the knowledge that can be extracted from the documents and their DTDs only. To collect this knowledge, which is used, e.g., to choose default formatting and to recognize useful fragments (see below), the document elements are first *classified* by mapping each element to some generic element [2]. For instance, all the elements that contain plain text (e.g., SGML #PCDATA) only and are of length less than 80 characters are mapped to elements called *String*. In the same way, longer elements which contain *String* elements or plain text, are mapped to elements called *Paragraph*.

In addition to classification, the documents are divided into *fragments*. A fragment is a coherent, contiguous, and relatively independent part of a document. To get best results for document reuse each fragment, like a chapter or a section, should be initially authored to discuss a well-defined topic. However, it is often not possible to manually re-structure an existing document base. To enable fragment-based assembly from existing document collections we have developed automatic statistical methods for the recognition and markup of probably useful document fragments [4, 3]. Fragmentation uses the results of classifying document contents using generic elements: the lowest-level fragments are usually sequences of elements classified as generic *Paragraph*s. Additionally, the fragment hierarchy consists of the ancestor elements of these minimal fragments.

Preprocessing, or the earlier writing phase, may also include adding supplementary information that can guide the assembly. Adding intelligence to documents is called

seeding [9, 1]. We have been experimenting with a manually seeded document collection of three text books on control engineering. The contents of the books have been classified by the authors according to the educational level and the nature of the content. The classification is attached to the corresponding document elements as SGML attributes. Information can also be added automatically. For instance, we have computed keyword lists for the document fragments.

In the first actual assembly phase, the user selects fragments to be included in the resulting document. Our system allows four ways to select elements. The first one is a rather standard search that includes 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 browsing *clusters*, which are collections of document fragments that are mutually similar within the cluster but dissimilar to fragments outside the cluster. Similarity of two fragments is measured in a standard fashion by calculating the proportion of their common key words. Clusters are used as a way to present large portions of the collection as browsable clumps of similar content. An interactive cluster based browsing approach known as Scatter/Gather was introduced by Cutting et al., who also presented efficient algorithms for it [7, 6]. Scatter/Gather browsing has been reported to be an effective way to utilize unfamiliar information collections in a way that is complementary to traditional query based retrieval of documents [16].

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

The fourth possibility is to start from the selected fragments and let the system search for similar or related fragments in the collection. Again, measuring the similarity of fragments can be based on comparing the sets of their key words.

Moreover, the assembly system allows the user to modify the set of selected document fragments in order to obtain the material of a final assembly. The modifications include discarding and rearranging the fragments, moving or copying selected fragments to be part of an existing fragment, and replacing a fragment either by its parent or children elements in the fragment hierarchy. Modifications can be controlled by a DTD that is constructed in the preprocessing phase. The classification gives a natural way to construct a simple generic DTD that expresses the hierarchy of generic elements only, e.g., that chapters may contain sections but not vice versa. As the result of the selection phase, we have an ordered list of document fragments.

Intermediate assemblies can be presented to the user, e.g., as HTML pages, by adorning each generic element with suitable formatting markup. However, the final assembly should be a valid and coherent SGML document, such that it can be further processed by other SGML applications. This may necessitate some restructuring, e.g., if we have a sequence of sections and one paragraph in the middle of them, we might need to create a new section to wrap the paragraph. Moreover, in order to be sensible, the assembly may need to be completed with additional elements. For instance, a sequence of paragraphs could be completed by a section title and an introductory para-

graph, or targets of cross-reference links could be included. The heuristics guiding these complements are based on the knowledge of the role of the generic elements.

One of the benefits of classifying the original elements by mapping them to generic elements is that we can attach semantics to the generic elements, e.g., how to format them. Hence, we are also able to format and print the resulting assemblies, achieving a uniform presentation for various elements from heterogeneous documents.

3 SAW — The SID Assembly Workbench

This section describes our prototype of a document assembly system, which we call the *SID Assembly Workbench*, or *SAW*. We have developed the system in order to test and to evaluate the conceptual model of an intelligent document assembly process, which we discussed above.

The assembly starts from a collection of document fragments. The user assembles a result document by iteratively selecting and reordering fragments from the collection. When the user is satisfied with the result, he/she saves it. If the result document contains fragments over several DTDs, the system may transform them to conform to a selected target DTD. SAW does not support actual editing of the assembly result. For producing a final deliverable such as a printed book, an on-line document or a CD-ROM, the result document can be further edited and formatted using an SGML editor or publishing system.

3.1 The SAW User Interface

The SAW user interface is based on the concept of a *view*, which is a representation of either the document collection or the result at some state of the assembly process. Conceptually, a view is an ordered sequence of assembly components, which can be document fragments, clusters of fragments, or other views. The SAW user interface consists of a result window that presents the view together with the commands that can be used for manipulating the result (Fig. 2). An initial view presents the entire document collection. The operations on a view always create a new view. By modifying the view, the user gradually moves towards a more final result consisting of a sequence of useful document fragments. The user can also store intermediate views, e.g., a draft assembly of a chapter. Stored views can be retrieved and included in new result views.

The user interface is divided into three logical parts. The first part presents a view of the document collection, also called the *current view* of the collection. The second part consists of the commands available for modifying the current view. They include selection and reordering of fragments, and clustering and searching in the collection, as well as advanced querying of the underlying database. The third part of the user interface provides a way of saving the results, i.e., the assembled views.

The current view consists of a table of rows, one row for each assembly component in the view. Figure 2 presents an initial view of a collection of three Finnish text books on fuzzy logic and control engineering. Each document element is presented as a short fragment summary in the current view. The fragment summary may include the name of the fragment (as is the case in Fig. 2) as well as the position of the fragment in the

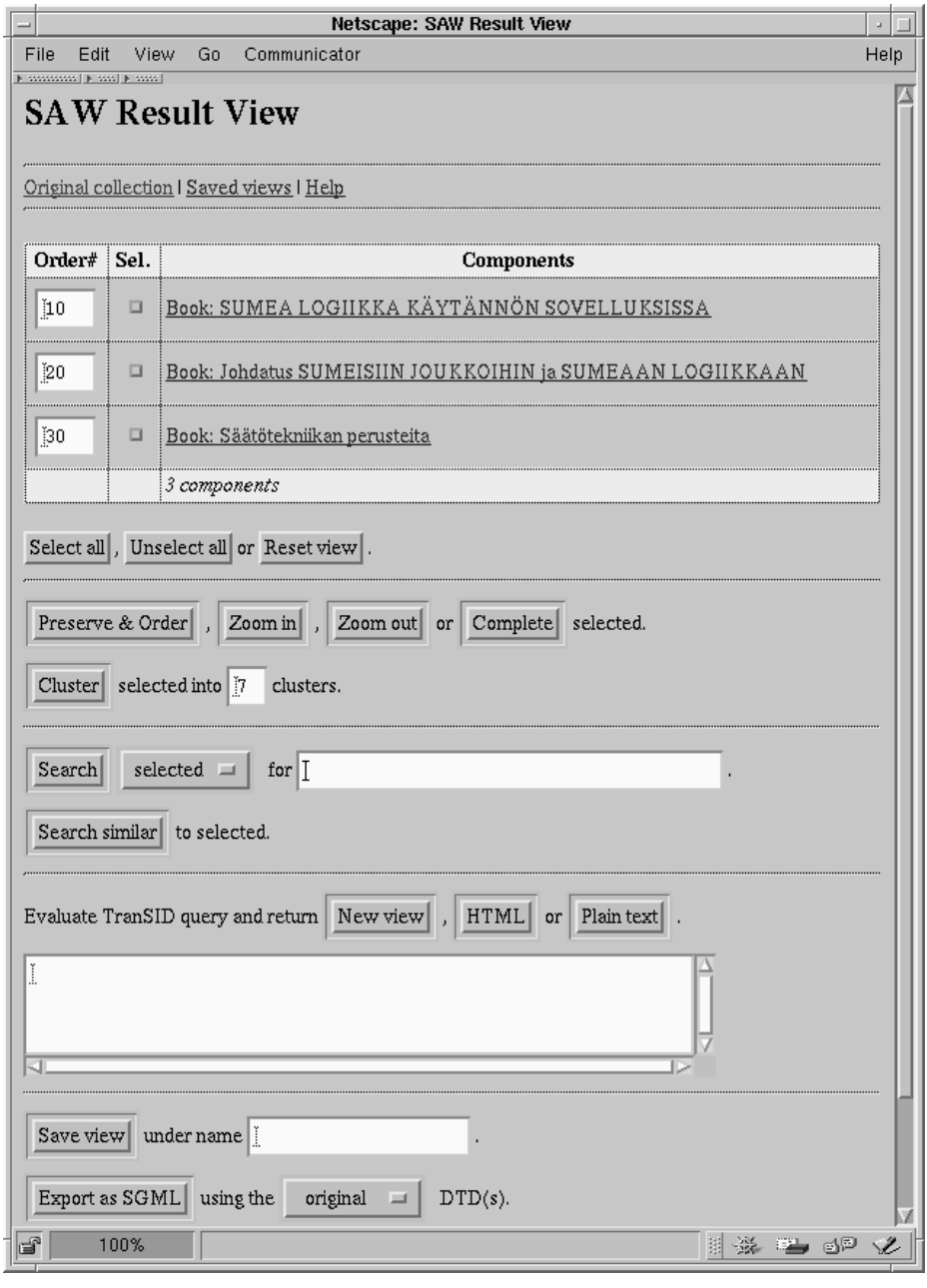

Fig. 2. The user interface of SAW.

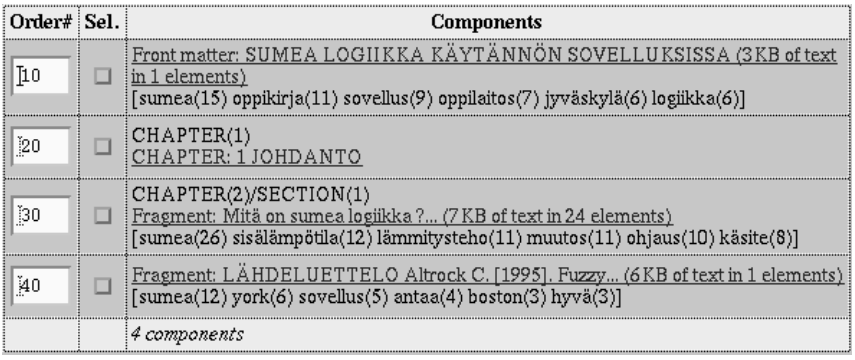

Fig. 3. Presentation of an intermediate view.

document structure, an indication of its size and a list of the most frequent keywords contained in the fragment (Fig. 3). For example, the third fragment (*Mitä on sumea logiikka?* . . .) is located in the first section of the second chapter, it has a size of 7 KB and it consists of 24 SGML elements. The two most frequent keywords of the fragment are *sumea* (fuzzy), appearing 26 times, and *sisälämpötila* (indoor temperature), appearing 12 times. By clicking on the fragment name, the user may view the textual contents of the fragment.

Each fragment is preceded in the current view by a field for an order number and a selection button. The order numbers can be changed to rearrange the view according to the numbering. Selections define which fragments are affected by the view manipulation commands. All fragments in the view can be selected or deselected at once using the buttons *Select all* and *Unselect all*. Any selecting or reordering of the current view can be undone using the button *Reset view*.

The *Preserve & Order* button discards all unselected fragments from the view, and arranges the remaining ones according to their order numbers. The *Zoom in* button replaces selected document fragments by their immediate subelements, e.g., tables by their rows. *Zoom out* correspondingly allows the user to navigate in the document structure to a higher level by replacing, say, selected section elements by their enclosing chapters. To make a consistent result document, we often want to include also chapter titles and introductory paragraphs that form the context of a chosen fragment. The *Complete* button includes fragments relevant to the selected ones in the current view.

The SAW system also allows the user to browse the contents of the collection as *clusters*, which gives the user an extracted view of the documents. Thus, the user can choose the appropriate clusters and fragments for re-clustering, and use these to search for appropriate fragments on new subsets of the document collection. The browsing mechanism employs the Scatter/Gather clustering techniques developed by Cutting et al. [7, 6].

The user may also *search* for keywords in the selected fragments, in the current view or in the entire fragment collection. Fragments that contain the specified keywords are returned in a new view. The user may also search for fragments *similar* to the selected

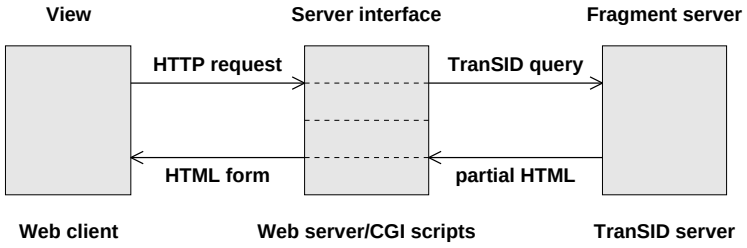

Fig. 4. SAW architecture.

ones in the current view. Advanced users may also formulate powerful queries using the TranSID query language [12] of the underlying database server. The result of an advanced query may be presented as a new view, an HTML page or plain text.

Figure 3 shows an intermediate (or possibly final) view of the document collection. The user has performed the following actions. He/she has selected and zoomed in on the first book, then selected the front matter, the two first chapters and the reference list (*LÄHDELUETTELO*), and finished by zooming in on the second chapter. This very simple view presents a digest of the book that could be used, e.g., as a marketing sample of the whole book.

When the user is satisfied with the current view he/she may save it for later use. The current view may also be exported as SGML fragments; exporting may include transformation of fragments so that they conform to a certain selected DTD. Fragments can also be stored unmodified. During the assembly process, the user may return to the original view of the collection, or to the saved views of the current session. The previous view may always be reached by clicking the *Back* button in the browser.

3.2 Architecture and Implementation

The SID Assembly Workbench is implemented as a Web server. The architecture consists of a user interface running in a Web browser, a server interface, and a fragment server (Fig. 4).

The view interface is realized as an HTML form, which together with the browser constitutes the client side of the SAW architecture. The form is constructed from a constant part, which contains the command buttons, and a dynamic part, which presents the contents of the current view. A hidden numeric identifier is attached to each fragment, cluster or view shown in the current view. The identifiers connect the assembly components of the current view to the corresponding document structures maintained by the fragment server. The state of the assembly process is maintained mainly by these identifiers in the view form. View modifications are realized by sending information about the chosen action and the contents of the current view to the Web server using the HTTP post method. The server interface scripts parse the request and pass it as a query to the fragment server. The response of the fragment server is passed back to the user as an HTML page.

The fragment server consists of a structured text database process. The structured text database is based on the internal data structures of an SGML transformation language called TranSID [12]. TranSID models documents as tree structures, and allows any part of the document tree to be accessed. The tree structures are maintained in main memory. The fragment server accepts queries expressed in the TranSID language, and responds by returning plain text, HTML, or partial HTML to be presented as a new view.

We chose the Web/HTML-based implementation of the user interface for several reasons. A Web browser offers a familiar look-and-feel and provides many of the user interface actions for free. HTML pages are easy to redesign, which makes the maintenance and tailoring of the system easier. This allows tailored initial views to be offered for different user groups. Also, an HTML interface is ready to be used over an intranet or to be integrated in a Web site on the Internet. On the other hand, certain operations are less flexible with a form-based user interface than they would be with a dedicated direct manipulation interface. For example, ordering of fragments is less natural than it would be in a drag-and-drop environment. Anyhow, this is not an inherent limitation of the chosen architecture. A more elaborate implementation of the user interface, e.g., based on Java applets, would allow manipulation of the current view in a more direct fashion.

We have tested the system with a collection of three text books and 3,125 statutes. The text books together with their keyword lists comprise about 660 kilobytes of text and 47,000 SGML elements. The size of the statute collection is about 12 megabytes, or 327,000 SGML elements. The fragment server loads document structures in main memory only when they are accessed, which implies that it takes only a few seconds to start the server to access the collection.

Currently, all the user operations mentioned in the previous section have been implemented, except for the completion of the result fragments. Also, assembled documents can currently be exported in HTML form only. The classification and fragmentation actions have been performed outside the current SAW implementation, but we expect to integrate these preprocessing actions in the workbench as well.

4 Conclusion

We have presented a model and an implementation of a system for intelligent document assembly. The system uses a database of SGML documents from which the user assembles new documents. The assembly is based on document fragments: the user chooses among document parts and selects appropriate fragments to be included in a new document. The assembly system supports browsing and reorganizing of the fragments as well as some more sophisticated techniques such as cluster-based browsing and structured search. The system is in a prototype phase and we are just beginning to evaluate the usefulness of our assembly model. We expect that further prototyping with document assemblies will reveal some real challenges, like managing explicit and implicit dependencies between document fragments.

We consider document assembly in a large collection primarily as an in-house publishing activity. If an assembly interface is offered to the open public, concerns of copy-

right and security have to be taken into account. Solutions to the copyright and security problem have been suggested in more restricted library environments [5], but imposing restrictions on the use of the document material in a more liberal document assembly framework like ours remains an open issue.

References

1. H. Ahonen, B. Heikkinen, O. Heinonen, J. Jaakkola, P. Kilpeläinen, G. Lindén, and H. Mannila. Intelligent assembly of structured documents. Technical Report C-1996-40, University of Helsinki, Department of Computer Science, June 1996.
2. H. Ahonen, B. Heikkinen, O. Heinonen, and P. Kilpeläinen. Assembling documents from digital libraries. In A. Hameurlain and A. M. Tjoa, editors, *Proceedings of the 8th International Conference on Database and Expert Systems Applications, DEXA '97*, number 1308 in Lecture Notes in Computer Science, pages 419–429, Toulouse, France, Sept. 1997. Springer-Verlag.
3. H. Ahonen, B. Heikkinen, O. Heinonen, and M. Klemettinen. Discovery of reasonably-sized fragments using inter-paragraph similarities. Technical Report C-1997-67, University of Helsinki, Department of Computer Science, Nov. 1997.
4. H. Ahonen, B. Heikkinen, O. Heinonen, and M. Klemettinen. Improving the accessibility of SGML documents – A content-analytical approach. In *SGML Europe '97*, pages 321–327, Barcelona, Spain, May 1997. Graphic Communications Association.
5. L. C. Anderson and J. B. Lotspiech. Rights management and security in the electronic library. Research Report RJ 9977 (89065), IBM Almaden Research Center, Aug. 1995.
6. D. R. Cutting, D. R. Karger, and J. O. Pedersen. Constant interaction-time Scatter/Gather browsing of very large document collections. In R. Korfhage, E. Rasmussen, and P. Willett, editors, *Proceedings of the 16th ACM SIGIR Conference*, pages 126–134, Pittsburgh, PA, USA, June 1993.
7. D. R. Cutting, J. O. Pedersen, D. Karger, and J. W. Tukey. Scatter/Gather: A cluster-based approach to browsing large document collections. In N. Belkin, P. Ingwersen, and A. Mark Pejtersen, editors, *Proceedings of the 15th ACM SIGIR Conference*, pages 318–329, Copenhagen, Denmark, June 1992.
8. H. Davis and J. Hey. Automatic extraction of hypermedia bundles from the digital library. In *Digital Libraries '95, Proceedings of the Second International Conference on the Theory and Practice of Digital Libraries*, Austin, Texas, USA, June 1995. Available at `http://www.csdl.tamu.edu/DL95/papers/davis/davis.html`.
9. P. M. English and R. Tenneti. Interleaf active documents. *Electronic Publishing – Origination, Dissemination and Design*, 7(2):75–87, June 1994.
10. C. F. Goldfarb. *The SGML Handbook*. Oxford University Press, 1990.
11. ISO. *Information Processing — Text and Office Systems — Standard Generalized Markup Language (SGML), ISO 8879*, 1986.
12. J. Jaakkola, P. Kilpeläinen, and G. Lindén. TranSID: An SGML tree transformation language. In J. Paakki, editor, *Proceedings of the Fifth Symposium on Programming Languages and Software Tools*, pages 72–83, Jyväskylä, Finland, June 1997. Technical Report C-1997-37, University of Helsinki, Department of Computer Science.
13. W. E. Kimber. Re-usable SGML: Why I demand SUBDOC. In *SGML '96*, Boston, MA, USA, Nov. 1996. Graphic Communications Association.
14. D. M. Levy. Document reuse and document systems. *Electronic Publishing – Origination, Dissemination and Design*, 6(4):339–348, Dec. 1993.

15. J. McFadden. Hybrid distributed database (HDDB) and the future of SGML. In *SGML Europe '96*, pages 321–327, Munich, Germany, May 1996. Graphic Communications Association.
16. P. Pirolli, P. Schank, M. Hearst, and C. Diehl. Scatter/Gather browsing communicates the topic structure of a very large text collection. In *Proceedings of the Conference on Human Factors in Computing Systems, CHI '96*, pages 213–220, Vancouver, British Columbia, Canada, Apr. 1996. ACM. Available at `http://www.soe.berkeley.edu/~schank/parc/pp_txt.htm`.
17. D. Stribling, T. Hunter, L. Olszewski, A. Corrigan, R. Mullis, and L. Allen. A real world conversion to SGML. In *Proceedings of the 14th Annual ACM Conference on Systems Documentation, SIGDOC '96*, pages 75–86, Research Triangle Park, North Carolina, USA, Oct. 1996. ACM.