ELECTRONIC PUBLISHING, VOL. 2(4), 211–229 (DECEMBER 1989)

Automatically transforming regularly structured linear documents into Hypertext[1]

RICHARD FURUTA, CATHERINE PLAISANT, AND BEN SHNEIDERMAN

Human Computer Interaction Laboratory and Department of Computer Science
University of Maryland
College Park, MD 20742
USA

SUMMARY

Fully automatic conversion of a paper-based document into hypertext can be achieved in many cases if the original document is naturally partitioned into a collection of small-sized pieces that are unambiguously and consistently structured. We describe the methodology that we have used successfully to design and implement several straightforward conversions from the original document's machine-readable markup.

KEY WORDS Hypertext conversion Document structure Conversion methodology

1 INTRODUCTION

As hypertext systems become widely available and their popularity increases, attention has turned to converting existing collections of information into hypertextual form. Many such projects have focused on reuse of previously developed documents; for example Egan *et al.*'s SuperBook [1,2], which automatically provides additional means for traversing the paper document's interactive display, Nunn *et al.*'s conversion of the IBM REXX manual [3],[2] and Glushko's [4] conversion of an encyclopedia into hypertext, a project that emphasized the importance of careful design of the resulting hypertext. Like paper documents, creating a successful hypertext requires care and creativity in its design and specification. A poorly-thought-out hypertext can be quite unattractive—indeed in such cases the reader may prefer a more static representation. In many cases the sources must be restructured to take advantage of the hypertext environment [5]. A commercial product, Texas Instrument's HyperTRANS, provides the means for iteratively specifying the conversion from paper-based document representations to hypertext, and Mamrak *et al.*'s experience in the Chameleon project [6,7] in recognizing paper-based document structures and converting among such document representations (with manual assistance) is also of relevance.

Conversions such as these address the issues in general conversion of relatively freely structured paper documents, for example books, articles, and monographs. Also of

[1] We appreciate the support of NCR Corporation in providing partial funding for this research.
[2] Based on the experience of manually carrying out the REXX translation, Nunn *et al.*, also reconverted part of the manual in a semi-automatic conversion.

0894–3982/89/040211–19$9.50

Received 6 November 1987
Revised 13 November 1987

1. Design the structure of the target hypertext article
2. Determine how the source's structure corresponds to the desired target's structure.
3. Specify the conversion process, which must
   (a) Extract the relevant components of the source's structure
   (b) Reorganize the components to form the target's structure
   (c) Augment the target's structure with representation of relationships (links)
   (d) Generate the hypertext database files
4. Automatically convert from source to produce the target hypertext database.
5. Modify the hypertext database, if appropriate, to provide a "wrapper", to incorporate additional articles, and to correct errors generated in conversion or carried forward from original source.

Figure 1. Transformation methodology (modified from reference 9)

importance are conversions that focus on more finely structured collections of information such as catalog entries, dictionary listings, and databases. (See, for example, the grammar-based techniques developed by the Oxford English Dictionary project [8].)

In a companion report to this one [9], we have described and evaluated our own experiences in converting preexisting collections of information into hypertext. In that report we characterize the steps we carried out in defining and executing the conversion (the steps are shown in Figure 1). These steps can be divided into those that are carried out manually (e.g., design of the hypertext) and those that can be carried out automatically. For information sources such as scientific documents, our experience suggests that the manual steps are most appropriately carried out for each "instance"—the transformation steps taken are highly specialized for the particular document being converted. Information sources exhibiting a more regular structure can often be converted as a class instead of as individuals. Although manual steps must be carefully carried out to produce a usable design, the resulting design can be applied without further change.

One example of such an information source is the cards in a library's card catalog. The catalog as a whole contains many relatively short cards, each card is regularly structured by predefined rules [10,11], and consequently the cross-references between the pieces can be detected mechanically.

A hypertext database corresponding to the catalog might have a separate entry for each card. The information contained in the entry would be the same as that contained on the card, but the information might be reordered to take advantage of the electronic presentation. Separately generated indices could provide access to the cards alphabetically

by author, title, and subject, while relationships between related books could further be represented by links from entry to entry.

The specification of the transformation from card to hypertext entry for one card is quite likely to be applicable to the other cards. Specification of the interrelationships among entries similarly follows a regular pattern.

Earlier, Shneiderman identified three "golden rules of hypertext" [12]:

- there is a large body of information organized into numerous fragments,
- the fragments relate to each other, and
- the user needs only a small fraction at any time.

Completely automatic transformation of a class of information sources becomes feasible when the source follows the "golden rules", and also is formed from regularly structured fragments whose boundaries may be detected unambiguously.

In this report, we will describe at length the organization of the systems that we have used in fully automatic transformation of two such regularly and consistently structured fragmented information sources. The organization of these systems follows the same general system architecture (this is step 4 in Figure 1's list), which will be presented in Section 3. The development of the transformation specification from source to hypertext is centered around careful analysis of the structure of both the source and of the target. In our converters, we first detect the *logical components* of the input that correspond to components in the hypertext. We describe the form of the logical components with a template or simple grammar, extract the components with a parsing program, and finally automatically create the hypertext database, rearranging the ordering of the logical components when appropriate. The designs of two transformations based on the architecture are described in Section 4.

Creation of a hypertext involves two separate but interrelated activities: (1) generation of the individual articles in the hypertext's database, each with their own internal structure and (2) identification of the interrelationships among the separate articles (forming the contextual links). While our description of the architecture focuses on structural transformation, the identification of logical components is also of use in detecting the interrelationships among informational elements, as such interrelationships are reflected in the partitioning of the source into logical components.

Our target hypertext system has been Hyperties.[3] Because specification of the transformation requires understanding of both the structure of the source and the structure of the target,[4] we will present much of the remaining discussion in Hyperties-specific terms. However, we wish to emphasize that the framework is general enough to be applied to many hypertext systems.

[3] Hyperties was developed in our lab, and our experience with it dates back to 1983. The IBM-PC version of the Hyperties browsing and authoring tools are now commercially available from Cognetics Corporation, and research in our lab continues, based on a Sun version. The work described here targeted the commercial version of the system.

[4] See Section 2 for a brief review of the relevant structural aspects of paper-based documents and of the Hyperties database.

Our major conclusion from this activity is also generally applicable. Our experience has convinced us that it is wise to separate the concerns accompanying full-scale hypertext conversion of arbitrary information sources from those concerns associated with smaller-scale specialized conversions. As the specific transformation examples shown in this paper will demonstrate, fully automatic conversion of medium-grained information sources that are regularly and consistently structured can be quite manageable. On the other hand, conversion of larger, more complexly and arbitrarily structured documents, such as scientific papers, seems to be a process that inherently requires manual assistance, even when considering only structural conversion,[5] if for no other reason than to provide the initial partitioning of otherwise undivided text into hypertextual units and to resolve ambiguous language use in determining links. Fully automatic specialized conversions can be achieved, even in the absence of complete solutions to full-scale generalized conversion.

2 DOCUMENT STRUCTURES

Before we can design a transformation from a paper-based document to a hypertext, we must first understand the data representations of the source and of the target. In this section, we sketch the characteristics of these two representations, first considering the desired representation of the source, as a structured document, and then describing the representation of the targeted Hyperties database.

2.1 Structured documents

The transformations that have been defined have started with the assumption that the paper-based document is described by (or can be converted into) a "structured document" representation [13]. The structured document representation identifies the logical components of the document, separating out the specification of the physical placement of the components onto the page. The representation is object-based—higher level objects are formed by composing lower-level objects. As example, a book might be defined as a sequence of chapters, each chapter as a sequence of sections, with the decomposition continuing until the document is described in terms of "primitive" objects—perhaps individual characters in this example.

Paper-based document preparation systems generally accept an author-prepared *marked-up document*—the content of the document interspersed with "markup elements" that identify objects within the document and provide other information to the computer program (the formatter) that is responsible for converting the marked-up document into and representation suitable for printing on the page. Identifying the structural components of the paper-based document is simplified if the marked-up document is available—indeed, identifying structural components in a scanned document remains an active research topic. Markup languages come in two flavors: those that describe the document's structural components (e.g., Scribe, LaTeX, and SGML) and those that describe the document's

[5] One justification for this claim is suggested by the Chameleon project's demonstration that conversion from arbitrary markup into SGML is ambiguous [7].

physical appearance (e.g., Runoff, Script, and *troff*). When a physically oriented markup language has been used, we must extract the logical elements. This can be difficult if the document's author has not been consistent in use of the markup commands.

2.2 The Hyperties database

The overall model of a Hyperties database, presented both to an author and to a reader, is based on the metaphor of an "electronic encyclopedia". From this external point of view, the database consists of a set of short articles related to one another by links, which are represented as user-selectable highlighted strings within the body of the source article.[6] Each article is further subdivided into three fields: the title, a short string naming the article; the definition, a short phrase describing the article; and the body of the article. The display of an article shows both the title and the body. When a user selects a link, the destination article's title and short description are shown in a separate window. Confirming the selection causes the source article's display to be replaced by the destination article.

The full PC-based Hyperties system permits both textual and graphical article bodies, and has many other features. A global index, listing all articles found in the database, is accessible to the reader at all times who can use it to select articles of interest. Fast string search can be enabled over the bodies of the articles, greatly enhancing the usability of converted databases. It is worth noting, that the database access structures generated by the mechanisms presented in this paper (for example, additional indices) are intended to complement the facilities already found in Hyperties.

The Hyperties database is represented as a collection of files. Two file types are of particular interest to us—those containing the articles (`FIL` files) and the one containing the global index (`INDEX.TIE`). A detailed description of the format of the article and index files may be found in Appendix A. Here we give an overview of the characteristics of interest in this report.

Each of the articles is stored in a separate file. The file is subdivided into the article's title, its short description, and its body, with imbedded links delimited with the ~ character. The target of a link is another article, and as might be expected, the identity of those targets (the filename containing the target article) is also associated with the source article. In addition to these fields, the article's file format contains delimiters and offsets to aid in locating the fields.

The global index contains an entry for each article, listing the article's index title and synonyms. The index title is usually the same as the article title, just mentioned, but it is possible that the two may differ. The synonyms are the strings that have been used to refer to the article—in other words the text of the imbedded links found within the article bodies.

[6] As can be seen from this description, the primary unit of interest to the Hyperties reader is a relatively discretely sized element of information. The hypertext model favored by some other hypertext systems incorporates longer scrolls of information with facilities provided for targeting links within the information and for holophrasting. Guide [14] is a good example of such a system, and its model more closely corresponds to that of a structured documents than does Hyperties'. While an appropriate design for a converted hypertext might differ based on the capabilities of the target system, the characteristics of the task of extracting the structure from the original document and of specifying the needed structural transformations would be unchanged.

3 ARCHITECTURE OF THE AUTOMATIC CONVERSION

In this section, we turn our attention to the specification of the conversion. Figure 2 sketches out the architecture that we have used in designing several conversions. As the figure indicates, the document undergoes several transformations in conversion from paper-based input markup to Hyperties database. In the figure, each of the document's representations is shown as an oval and a transformation from one representation to another is shown as one or more arcs between the corresponding ovals, with the name of the translator shown adjacent to the arc(s).

The transformation of a source document into a Hyperties database is carried out in two steps. The first step, performed by the "domain-specific translator" in the figure, is specialized for each particular class of conversion. The form of the translator (to be described in more detail in Section 3.1) depends on the form of the input and on the design characteristics of the desired target Hyperties database. The result of this transformation is a linear representation of significant portions of the Hyperties article file representations (see Section 3.2 andAppendix B).

The second step, performed by the "linearized Hyperties DB to DB translator", is the same for all transformations. This transformation generates the missing parts of the Hyperties article files, collects the necessary information for the global index and then creates the necessary set of files that make up the Hyperties database. The resulting files can be read using the Hyperties browser tool and modified by the Hyperties author tool.

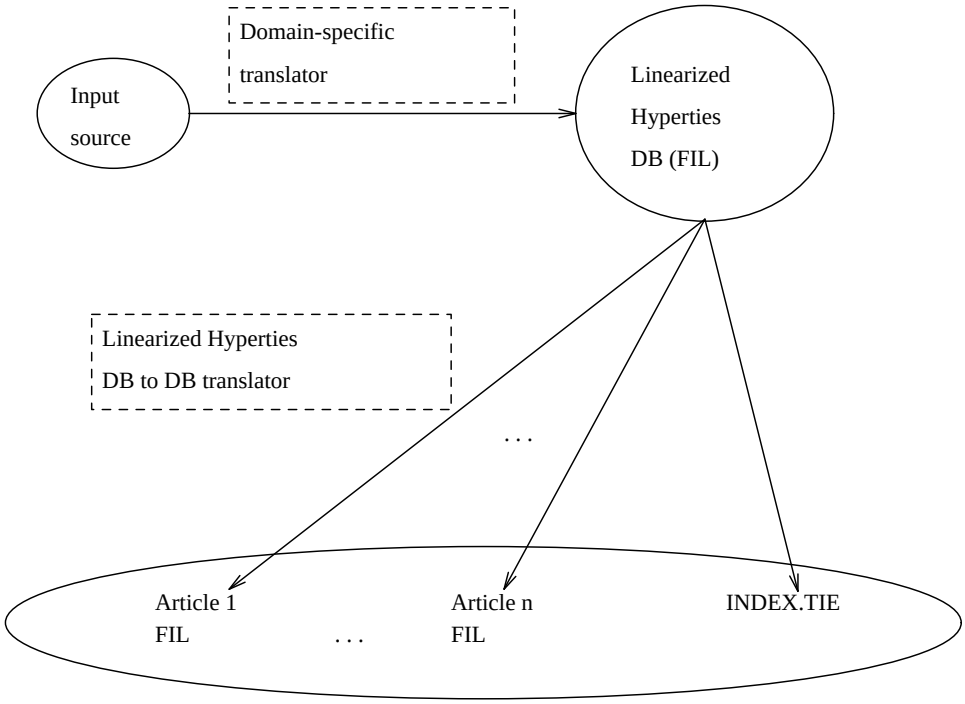


Figure 2. Overview of the conversion process

3.1 Domain-specific translation

Domain-specific translation involves recognition of the source's components, reorganization of the components to form the target representation, and augmentation of the target with links. Carrying out this step has required specification of the following components:

- A parser that recognizes the relevant parts of the input markup representation, resulting in a representation of the input as a logically structured collection of components. The parser's complexity varies greatly with the complexity of the input markup language and with the degree of similarity between the objects defined in the input markup language and the form of the desired logical components.
- Filters that can be applied to the content associated with a component. Filters may, for example, be needed to interpret or remove imbedded formatting commands. Another common use of a filter is to add those formatting commands defined in Hyperties' markup language to the content of the component.
- A function that generates the unique identifier that will serve as the eventual designator of an associated Hyperties article (i.e., the *filename* in implementation). The unique identifier is treated as if it were a document component, allowing its use in the mappings that will be defined subsequently.
- A mapping from (possibly filtered) components to an internal representation of the database. This internal representation will provide an abstraction of the database as a whole, which can then be used as the target of later specifications.
- Mappings from (possibly filtered) components to external representations. In this case, the content of selected components can be copied to one or more external files. These files may be then subjected to separate processing to produce, for example, one or more index lists. External representation may be employed for convenience (for example, to permit the use of the system's sorting routines), or by necessity (for example when conveying information between processing passes, as described below).

The result of these mappings is an internal representation of the document and zero or more external representations. These representations may be manipulated further and the linearized Hyperties database will then be generated. Manipulation of the representations and generation of the database encoding is carried out through use of some additional functions:

- "Threading functions" that can be applied to the internal representation connecting together some subset of the database's components. The threads are generally set based on some attribute of the component's content. These threads can then be used in generating links within the Hyperties representation and in generating indices listing the components along a particular thread. The head of a thread also contains a unique identifier and a "print name". This information also can be used in generating the body of a Hyperties article.

- Mappings from external representations that modify the internal representation. This permits use of externally manipulated data within the database. Such data may have been generated, for example, from information exported on a previous pass through the input.
- Mappings from the internal representation of the database to produce additional external database representations, e.g., the linearized Hyperties database. Generally, what is specified is a traversal of the internal representation and a specification of how the encountered components should be represented in the linearized database. Literal strings and thread-based links may be included in the linearized database.

The domain-specific translation must be specified anew for each different class of database. To date, each translator in this class has been specified either as a C-language program or as a central C program whose output is postprocessed with other Unix-based routines. As possible, the structure of these separate translators has been regularized, and we have developed a collection of library routines to aid in specification of the individual components of the transformation. Section 4 describes the design of two instances of the the domain-specific translator in more detail.

3.2 Linearized Hyperties database

The output from a domain-specific translator is a linear representation of the "core" of a Hyperties database. The linear representation encodes only the content of the articles in the Hyperties database, and only the parts of those articles that cannot be derived from other parts of the specification; for example, the global index is not directly represented in the linear representation nor are the binary offset counts found at the beginning of the actual Hyperties article file. Appendix A describes the format of the files in the Hyperties database and Appendix B shows the elements of the linearized Hyperties database representation.

The linearized Hyperties database representation identifies the following components of each article: filename, article title, index title, short description, content, and the displayed-string/target-filename pair within the content that specifies a link. Converting the linearized form into an actual Hyperties database requires generation of the individual article files in the appropriate representation and requires collection of information for the global index and generation of the global index file.

4 APPLICATION OF THE ARCHITECTURE

In this section, we present two fully automatic conversions that were based on the architecture sketched out in the previous section. Issues in design of these conversions were discussed in the companion report [9]. Here, we briefly review the characteristics of this design, focusing on the application of the architecture.

4.1 UMIACS abstract listing

In this section, we describe a conversion from the University of Maryland Institute for Advanced Computer Studies (UMIACS) annual *troff*-produced listing of collected

abstracts from the year's technical reports.[7] A sample abstract may be seen in printed form as Figure 3. This technical report is issued jointly by UMIACS and by the Computer Science Department; hence it carries two identifying numbers. The abstract was described by the *troff* source of Figure 4, and the corresponding Hyperties article is shown in Figure 5. Here, the identifying numbers are used as the article's title, and the article's body is formed by rearranging the components of the original listing. The abstract's title is also used as the Hyperties article's short description.[8] The article's body is augmented with links to a global index and to indices for each of the issuing departments, as well as links directly

UMIACS–TR–88–4
CS–TR–1973

Title: Efficient Stochastic Gradient Learning Algorithm for Neural Network
Author: Y.C. Lee, Department of Physics and Astronomy

Efficient first and second order adaptive learning algorithms of stochastic gradient descent variety are described. Both algorithms can automatically adjust step sizes to achieve optimum convergence rates. Various theorems concerning the convergence properties of these algorithms are discussed.

Figure 3. A sample UMIACS abstract

```
.(l L
.b UMIACS\-TR\-88\-4
.b CS\-TR\-1973
.sp
Title:          Efficient Stochastic Gradient Learning Algorithm
                for Neural Network
Author:         Y.C. Lee, Department of Physics and Astronomy
.)l
.sp
.pp
Efficient first and second order adaptive learning algorithms of
stochastic gradient descent variety are described.  Both
algorithms can automatically adjust step sizes to achieve optimum
convergence rates.  Various theorems concerning the convergence
properties of these algorithms are discussed.
.sp
```

Figure 4. troff markup of sample UMIACS abstract

[7] The 1988 UMIACS abstract listing contained 102 abstracts. The *troff*-based input markup file was approximately 125 000 bytes in length.
[8] Consequently, the abstract's title is used both in the Hyperties article body and also in the short description.

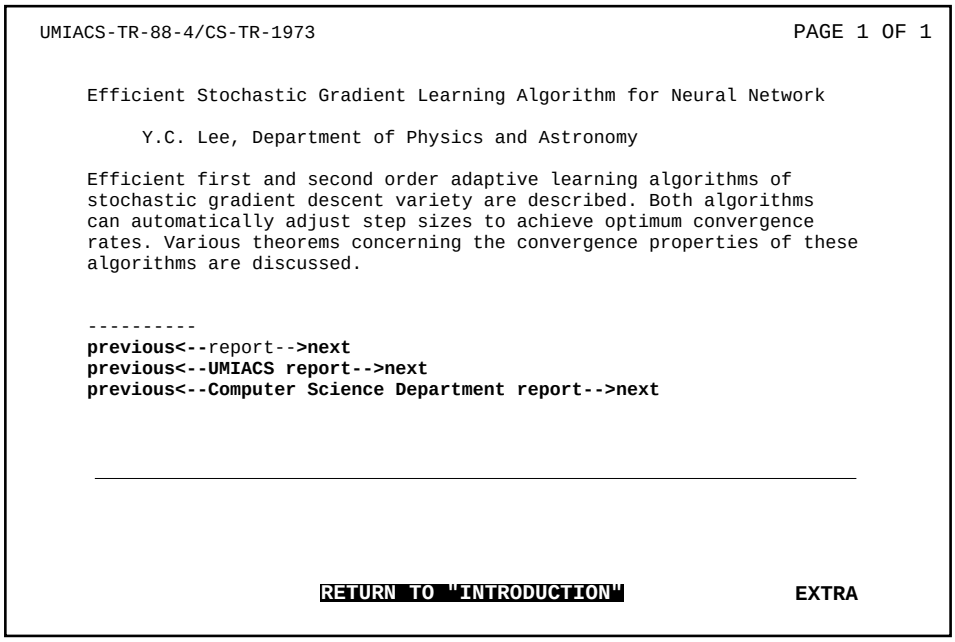


Figure 5. Hyperties article corresponding to sample UMIACS abstract

to the preceding and following abstracts in each of these indices. Each of these indices is automatically generated during the conversion.

Figure 6 shows the design of the domain-specific translator that converts the UMIACS abstract listing. The input source's markup in *troff* describes the appearance of the printed output, and not the logical structure of the document. Fortunately, the use of *troff* commands was relatively consistent within the markup, although, in a few cases, alternate descriptions were found producing output that was similar in appearance to the markup used in other portions of the input. We described the expected sequences of input commands through a series of regular expressions. The regular expressions were designed based both on the *troff* commands found in the input but also on phrases found within the text itself. Alternate markup command sequences were recognized and aliased to the primary sequence. The regular expressions were used to label the arcs of a deterministic finite automaton (DFA), shown in Figure 7.[9] The input stream was read on a line-by-line basis (reflecting the line-oriented command structure of *troff*). The input was then matched against the regular

[9] We were fortunate in that the beginning of each abstract began with a block of material in which line breaks were retained. This block was delimited by the *troff* `.(l` and `.)l` commands (these commands are defined in *troff*'s `-me` macro package). Handling the occasional source misencodings required added complexity in the translator, for example the transition in Figure 7 in state CH_AUT on "`.(l`" in addition to the expected transition on "`.)l`".

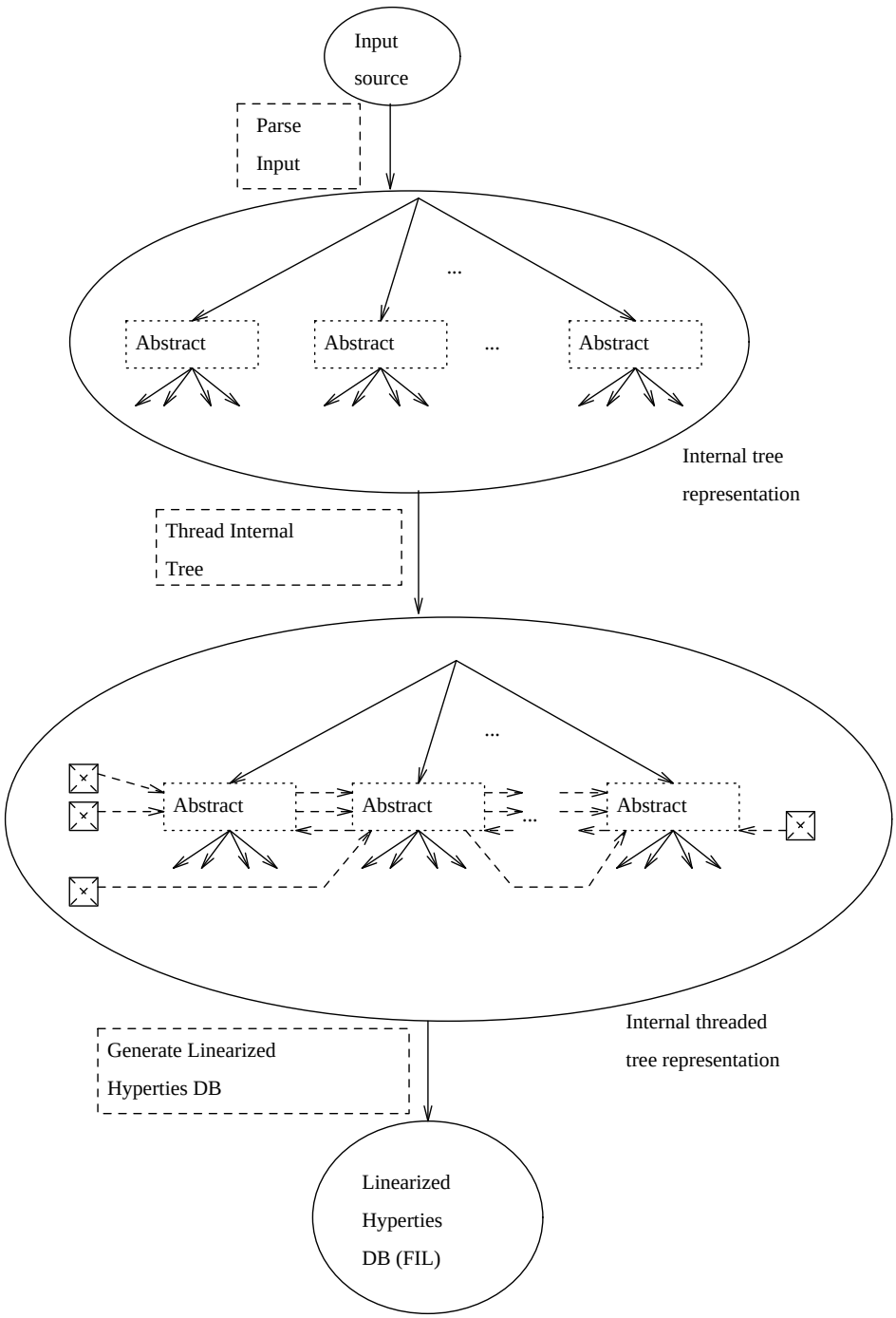
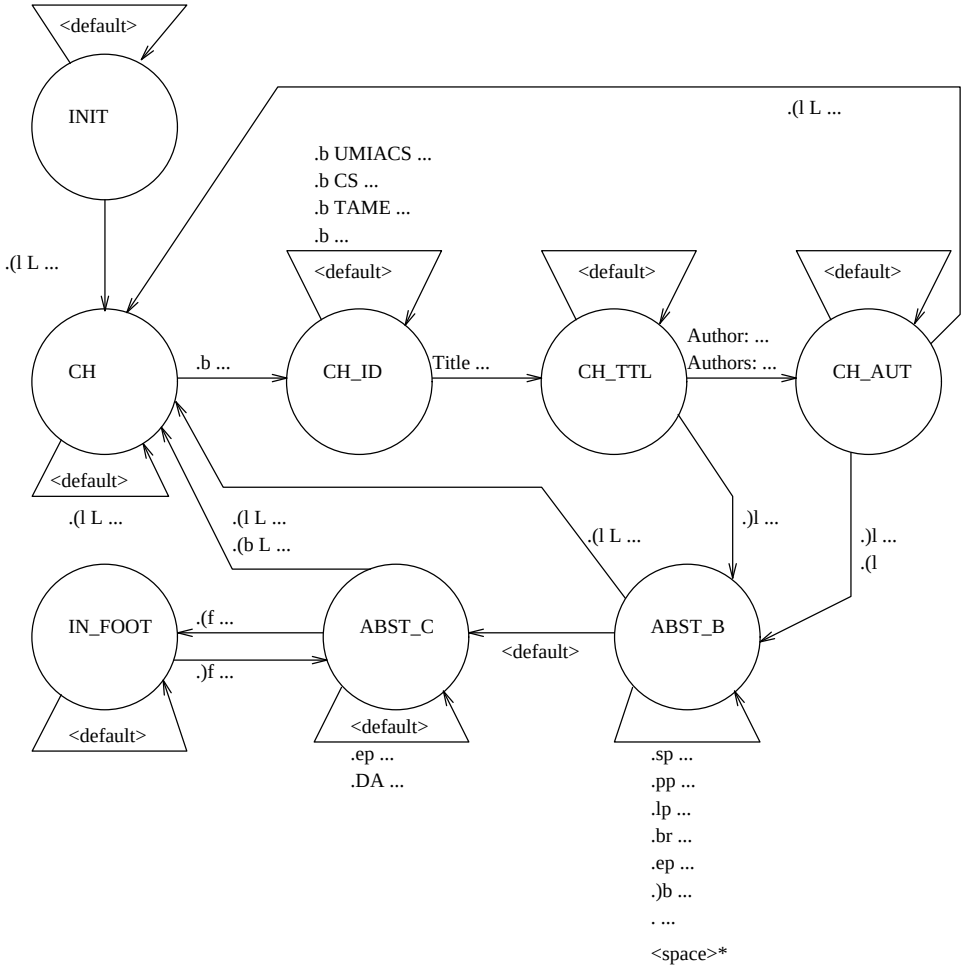


Figure 6. Domain-specific translator design for the UMIACS database



State	Description
INIT	initial state
CH	collect heading of abstract
CH_ID	collect id (TR numbers) within heading
CH_TTL	collect title of abstract
CH_AUT	collect author(s) of abstract
ABST_B	beginning abstract's text
ABST_C	collecting abstract's text
IN_FOOT	ignoring footnote within abstract

Figure 7. DFA controlling the translation

expressions on the arcs leading from the current state.[10] A default case was specified for each state, shown as <default> in the figure, and this default case was selected when no expression was matched. When a regular expression was matched, a specified routine was invoked on the input (the exit routine), the DFA's state was changed, and another routine invoked on the input (the entry routine). (These routines are not shown in the figure.) The job of the exit routine was frequently to close a data structure in the internal representation associated with the previous state while the purpose of the entry routine was to perform the necessary initializations for a data structure associated with the new state or to collect information to be inserted into an already initialized data structure. When either entry action or exit action was unneeded, for example when skipping a portion of the input, a null routine was specified.

The overall result of parsing the input was an internal tree representation of the entire database. The node corresponding to an abstract consisted of fields containing the generated filename that was to be associated with the corresponding Hyperties article, the list of the abstract's assigned technical report numbers, the abstract's title, its author list, and the body of the abstract.

The internal tree was then threaded, producing one thread that connected the articles sequentially, in order of definition, and one thread for each of the defined technical report series. (Three different series were defined in this listing: the previously mentioned UMIACS and CS reports, and a smaller sequence associated with the TAME software engineering project.) The threads are represented as dotted lines in Figure 6, with the head of each thread shown as a box at the beginning of the sequence of threads.

The internal threaded tree was traversed, and as abstracts were encountered the corresponding Hyperties article was generated. As each article was generated, links were formed by consulting the threads that ran through the corresponding abstract in the internal threaded tree representation. A separate index listing of reports associated with each of the defined report series was generated by traversing the associated thread list in its entirety, collecting and filtering the appropriate subfields as abstracts were encountered along the thread.

As a measure of the relative complexities of the different subtasks in this transformation, it is useful to more closely examine the distribution of source code to function. The total implementation required 3267 lines of C-language code. Of this, 1988 lines (61% of the total) went into the implementation of the domain-specific translator, 1023 (31%) lines went into the implementation of the linearized Hyperties DB to DB translator, and 256 (8%) lines into library routines and header files used by both (primarily string manipulation routines).

Figure 8 summarizes the distribution of code within the domain-specific translator. The entries in the table in the "problem-specific" column are specialized either for *troff* or for the specific domain of this translation. Those in the "general" column are easily reusable in other contexts as well. When the library functions are also taken into consideration, about half of the code in this translator is generic, rather than specific.

The linearized Hyperties DB to DB translator is used without change in other transformations, and indeed is useful in other applications as well (e.g., interchange of

[10] The trailing "..." in the figure's regular expressions matches zero or more arbitrary characters.

Activity	General percentage	Problem-specific percentage	Total percentage
DFA state definition	10	9	19
DFA execution	3		3
DFA execution actions		14	14
Pattern matching support	4		4
Thread management	17		17
Filters	2	11	13
Generate linear form	13		13
Guide specific translation		17	17
TOTAL	49	51	100

Figure 8. Distribution of code for UMIACS domain-specific translation

Hyperties databases). The primary task of this translator is regeneration of the portions of the Hyperties database's format missing from the linear representation, and the largest part of this regeneration (34% of the code) is creation of the index.

4.2 University Microfilms dissertation abstracts

A second conversion transformed a collection of 98 University Microfilms dissertation abstracts, dating from 1985 through the first part of 1989. The design of the Hyperties database (created by Ben Shneiderman and John Kohl) incorporated indices that permitted access by topics, author names, university names, and department or discipline names.

The input markup file had been specified in a custom-designed logical-object-oriented language, as shown in Figure 9. In markup, fields are identified by a two-letter code in the margin. Continuation lines are indented. The two-letter codes are straightforward—AN is the abstract's order number, AU for author, IN for institution, TI for title, SO containing a further cross-reference, DE for the issuing academic department, and AB for the body of the abstract.

The structure of the domain-specific translator for this dissertation database is shown in Figure 10. The structure of the translator differs from that of the UMIACS conversion

```
AN University Microfilms Order Number ADG87-15461.
AU BROWN, MARC HARRY.
IN Brown University Ph.D. 1987, 179 pages.
TI Algorithm animation.
SO DAI V48(04), SecB, pp1095.
DE Computer Science.
AB     An algorithm animation environment is a means for
   exploring the dynamic behavior of algorithms that makes
   . . .
```

Figure 9. A portion of a sample dissertation abstract in markup form

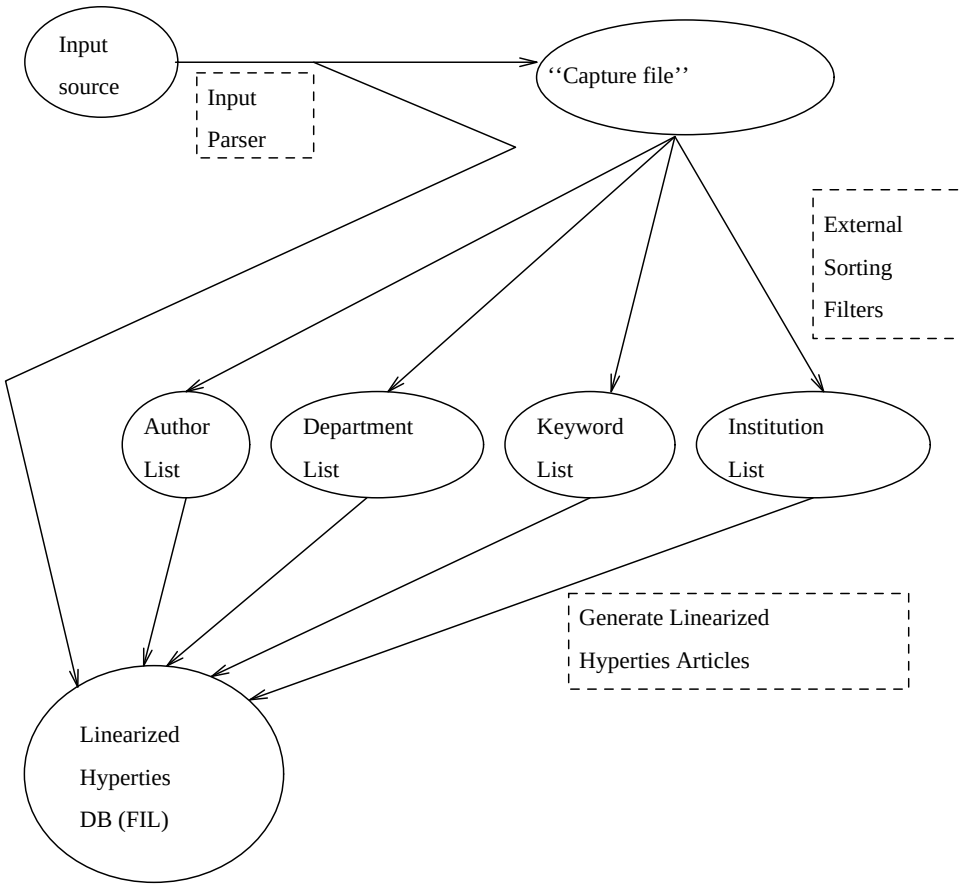


Figure 10. Domain-specific translator design for the dissertation database

(compare Figure 6), reflecting the different characteristics of the two input documents and the different features of the two Hyperties database designs. Because of the length of the dissertation abstracts (the input markup occupied about 235 000 bytes), the Hyperties articles (in linear form) were generated as the input source was parsed, and the content of the input was not retained in an internal data structure. Fields of the input representation were slightly reordered in the output Hyperties article, and some fields were slightly filtered (e.g., the dissertation title field was truncated to 60 characters and used as the Hyperties article's title).

The input parsing task was trivial to implement because of the regular structure of the input. The implementation of the input parser and associated filters totaled only 597 lines of C, and probably could have been made much smaller. Specification of the linearized form of the Hyperties database relied on the routines of the previous conversion to produce the linearized form, an additional 254 lines of code.

The implementation of the four index lists was based on summary information captured

during the transformation of the input. This summary information, stored in an external file, was sorted and filtered using Unix commands. The resulting sorted abstract lists were then converted into the linearized Hyperties database form by small (approximately 15 line) `awk` scripts and merged with the articles produced directly by the parser.

5 DISCUSSION AND CONCLUSIONS

We have demonstrated a framework that can be used to guide the fully automatic transformation of regularly and consistently structured fragmented information sources. While such information sources are only a subset of all information sources, they are frequently found—for example many kinds of catalog listings, traditional databases, and computer programs. Furthermore, there is great interest in achieving transformation of these kinds of information sources into hypertext. Even when the reader's primary means of gaining access to the database is by specific search queries rather than by browsing, a hypertextual organization can help provide a better understanding of the context in which the information exists.

The success of an automatic transformation is tied to the degree to which there is a natural logical relationship between the components of the input information source and the generated hypertext. Successful implementation of the transformation is affected by the degree to which the components of the information source's representation are represented unambiguously. One observation is that there is a natural affinity between generic markup and logically defined components. However, generic markup is not *required* in the information source's representation. Since a document's logical structure is generally reflected in its appearance, consistently used physical markup descriptions permit extraction of logically defined components. Database representations incorporate an explicit identification of the components of each entry that can be used in transformation. Even in the absence of markup, carefully applied standards can insure that the logical components can be detected; for example the previously mentioned cataloging rules of the American Library Association [10,11].

Just as logical structure can be represented in the absence of generic markup, it is also worth noting that generic markup need not necessarily reflect logical structure. A poorly-thought-out SGML Document Type Definition, for example, may provide little assistance for transformation if unrelated cases are merged or if the markup units do not correspond to natural logical divisions. From the standpoint of the translation developer, a clear understanding of the structure of the information source is as important in specification as is understanding of the target hypertext.

As noted, computer programs fall within the class of regularly and consistently structured fragmented information sources. Syntactically, programs exhibit a structure that is finer-grained and more highly constrained than that associated with printed documents. We expect that these structural characteristics can be used to advantage and that transformations defined for computer programs will be stated for the language (i.e., all programs written in that particular language) rather than for particular collections of instances, as is the case with document transformations.

Based on our experiences with the specific architecture and implementations described in this report, we are beginning to design a generalized toolkit to simplify the specification of a transformation. The general routines identified in the implementations described here form the basis for such a toolkit.

Finally, we have been pleasantly surprised by the significant amount of flexibility gained through the architectural decision to first produce the hypertext in linearized form, rather than to generate the hypertext database directly. Some of this flexibility results because the intermediate form permits a clean separation of the generation of the hypertext's content, relationships, and structure from the details of generating the specific file representations of the hypertext components. That the intermediate representation is formed from a string of printable characters also increases flexibility, as the representation can be generated from many separate sources, and indeed can be postprocessed easily with general-purpose utilities if necessary. For example, it would be easier to apply an external spelling corrector to the intermediate form than to the actual database.

The intermediate form was originally defined to aid in database transfer, as the transformations have generally been carried out in a computer environment (Unix) that is dissimilar from the environment that Hyperties runs in (IBM-PC). It may provide a convenient intermediary for interchange among different versions of Hyperties and for interchange between other hypertext systems and Hyperties.

REFERENCES

1. Dennis E. Egan, Joel R. Remde, Louis M. Gomez, Thomas K. Landauer, Jennifer Eberhardt, and Carol C. Lochbaum, 'Formative design-evaluation of "SuperBook"', *ACM Transactions on Information Systems*, **7**(1), 30–57 (January 1989).
2. Dennis E. Egan, Joel R. Remde, Thomas K. Landauer, Carol C. Lochbaum, and Louis M. Gomez, 'Behavioral evaluation and analysis of a hypertext browser', in *CHI'89 Proceedings*, pp. 205–210 ACM, New York (May 1989).
3. Debbie Nunn, John Leggett, Craig Boyle, and David Hicks, 'The REXX project: A case study of automatic hypertext construction', Technical Report TAMU 88-021, Department of Computer Science, Texas A&M University (April 1988).
4. Robert J. Glushko, 'Transforming text into hypertext for a compact disc encyclopedia', in *CHI'89 Proceedings*, pp. 293–298 ACM, New York (May 1989).
5. Charles B. Kreitzberg and Ben Shneiderman, 'Restructuring knowledge for an electronic encyclopedia', in *Proceedings of the International Ergonomics Association's 10th Congress*, Sydney, Australia (August 1988).
6. S. A. Mamrak, M. J. Kaelbling, C. K. Nicholas, and M. Share, 'A software architecture for supporting the exchange of electronic manuscripts', *Communications of the ACM*, **30**(5), 408–414 (May 1987).
7. Sandra A. Mamrak, Michael J. Kaelbling, Charles K. Nicholas, and Michael Share, 'Chameleon: A system for solving the data-translation problem', *IEEE Transactions on Software Engineering*, **15**(9), 1090–1108 (September 1989).
8. Gaston H. Gonnet and Frank Wm. Tompa, 'Mind your grammar: A new approach to modelling text', Research Report CS-87-13, Department of Computer Science, University of Waterloo (March 1987).
9. Richard Furuta, Catherine Plaisant, and Ben Shneiderman, 'A spectrum of automatic hypertext constructions', *Hypermedia*, **1**(2), 179–195 (1989).
10. Michael Gorman, *The Concise AACR2*, American Library Association, 1981.

11. *Anglo-American Cataloguing Rules, 2nd Edition*, eds., Michael Gorman and Paul W. Winkler, American Library Association, 1988.
12. Ben Shneiderman, 'Reflections on authoring, editing, and managing hypertext', in *The Society of Text*, ed., E. Barrett, MIT Press, Cambridge, MA, (1989).
13. Richard Furuta, 'Concepts and models for structured documents', in *Structured Documents*, eds., Jacques André, Richard Furuta, and Vincent Quint, pp. 7–38, Cambridge University Press, Cambridge (1989).
14. P. J. Brown, 'Turning ideas into products: The Guide system', in *Proceedings of Hypertext'87*, pp. 33–40 (November 1987). Published by the Association for Computing Machinery, 1989.

APPENDIX A PC HYPERTIES FORMAT

In implementation, the PC-based Hyperties' database (version 2.35i) is represented as a collection of DOS files. The articles are contained in a set of files with extension `FIL`, one article per file. The collection of articles is augmented with a single index file of name `INDEX.TIE`. Other types of files are also present, but are not of as immediate relevance to our transformations.

Each of the article `FIL` files is further subdivided into the following fields:

1. The title of the article.
2. The short description.
3. The body's contents, with the strings that correspond to links nested within delimiter characters (the character ~ before and after the string). Note that the filename corresponding to the link's target is specified in a subsequent field, not within the body.
4. A list of link target filenames. The first target filename corresponds to the first link defined within the body, the second to the second, and so on.
5. A note field, left empty in the transformation.
6. An empty field that had significance in earlier versions of Hyperties.

The file begins with a set of offset counts that point to start of these fields, and each field is separated from the next by a delimiter character.

The global index file, `INDEX.TIE`, consists of a sequence of lines of the general form

Title | `FILENAME` | *Status bytes* | synonym | synonym | . . .

All lines but the first correspond to an article. (The first line is a template line, showing the maximum number of synonyms defined within the index.) The title field is not required to correspond to the title defined in the article, but it generally does. Synonyms are the other strings by which the article has been identified in the definition of links. The three status bytes distinguish the introductory article that is shown when first starting up the Hyperties browsing program, indicate whether the article is non-empty, and indicate whether it is actually present in the database.

APPENDIX B ELEMENTS OF THE LINEARIZED HYPERTIES DATABASE REPRESENTATION

This appendix describes the format of the linearized Hyperties database. Only the articles are represented in the linearized form—the global index is generated from the information contained here.

The linearized Hyperties database is represented as a sequence of commands, one command per line. The description of this appendix shows the nine categories of commands and illustrates the form of each. The order of the elements that encode a particular article, shown below, is the order in which the commands will appear in the encoding of the article. The database as a whole will be represented as a sequence of article encodings. The generally used content designators are used in the indicated places within the article encoding elements.

Elements that encode a particular article

- Begin article mark/filename ($|\textit{filename}| \leq 8$)
  b*filename*
- Index title for article
  i< string >
- Article title
  t< string >
- Article definition marker
  d
  $(<\text{string-set}> \mid <\text{newline}>)^{+}$
- Article content marker
  c
  $(<\text{string-set}> \mid <\text{link-specification}> \mid <\text{newline}>)^{*}$
- Article end marker
  e

Generally-used content designators

- < string-set >
  s< string >
- < newline >
  n
- < link-specification > ($|\textit{filename}| = 8$)
  l*filename*< string >

The < string-set > is used to represent strings that may be longer than one line in length and to represent strings that are to be interspersed with links in the article's body; the < newline > marks the end of the generated line. The < link-specification > generates a link that is represented in the Hyperties article's body by the specified < string > with the given *filename* as destination.