ELECTRONIC PUBLISHING, VOL. 8(2 & 3), 181–193 (JUNE & SEPTEMBER 1995)

Filtering structured documents in the SYNDOC environment

E. KUIKKA[1]

Department of Computer Science and Applied Mathematics
University of Kuopio
P.O.Box 1627, 70211 Kuopio, Finland

A. SALMINEN

Department of Computer Science and Information Systems
University of Jyväskylä
P.O.Box 35, 40351 Jyväskylä, Finland

SUMMARY

This paper describes the filtering approach for searching documents whose structure is defined by a grammar. The method is based on the theoretical model for defining filters to specify information interest of a user. It is employed to find documents in SYNDOC, a syntax-directed text processing system. The method is suitable, for example, for SGML and ODA documents. The user selects a grammar and indexes only documents for the selected grammar. A filter generated in a syntax-directed way using the grammar describes conditions for indexed documents integrating structure and content constraints. The user compares a filter with indexed documents and, either edits, browses or prints original documents using the selected output form. Indexed documents, filters and retrieved documents can be stored for further purposes.

KEY WORDS Structured document Filtering Context-free grammar Parse tree

1 INTRODUCTION

In filtering [1], the user describes in a profile or filter which documents she or he wants to retrieve. A filter describes a long-term interest of a person or a group of persons as opposed to queries in interactive information retrieval systems. The filtering is characteristically used to remove unnecessary information or to choose necessary information from an incoming data stream. Instead of applying the filtering to all documents in a data stream, it is common to restrict the amount of source documents. For example, in news systems the user can select certain news groups and filtering is applied to these groups. There is however a need for various classes of document filters and their flexible use in document management.

Nowadays users have lot of documents in files. Text processing, e-mail and news systems create files. Documents in the Internet are entirely based on files which are linked with hypertext links. In many of these files, the content consists of structured text. When documents are in files it is easy to update, delete and insert them, but searching for them is a problem. To handle this problem, it is possible to use string search methods which are based on regular expressions (for example, grep program in the unix operating system, sgrep [2] and cgrep [3]). They can find out lines or parts of ASCII files where a given character string exists in documents. Another option is to store documents in a centralized repository or database as, for example, in Dynatext [4] or Maestro [5,6]. They offer SQL-type query capability to retrieve documents or their parts, but the update of documents requires the update of the repository as a whole by means of reindexing. The distributed system developed in MULTOS project [7] for multimedia documents is based on the client-server paradigm and allows loosely coupled components to store documents. Different

[1] This work was done during the author’s stay in the University of Waterloo in Canada.

CCC 0894–3982/95/020181–13

Received 25 March 1996
Revised 27 March 1996

access methods for an SQL-type language may be used for formatted data and for the text files in various components. Abiteboul *et al.* [8,9] have introduced a method which allows translations to be made from files to an object-oriented database and inverse translations from the database to files. Documents can be updated and retrieved in the database and represented to the user as files. Consens and Milo [10,11] have described a framework which allows to access and manipulate structured data regardless of whether it resides in a database or in a file system. The surveys in [12,13,14] describe retrieval languages for structured text. In the languages, criteria for the documents to be retrieved are specified by expressions which integrate structure information with content conditions. For end users to express their information needs, expression-based queries are often too complicated.

The popularity of the QBE-type (Query-by-Example) graphical query languages of relational databases shows that the use of the table schemas as templates in queries helps the end users to specify their information needs [15,16]. An example of a graphical text retrieval language using a hierarchic structure description as a template is LQL [17]. LQL is designed, however, for a specific application: dictionary entries.

In [18] we have defined a model to describe information interests of a user for structured text in a template. Templates are created from the grammar of the text. The basis of the model is the text model introduced in [19,20] and more extensively described in [21]. The aim of this paper is to describe how the model was used to search documents in SYNDOC (SYNtax-directed DOCument processing system) [22]. SYNDOC uses a syntax-directed approach for the input, output, retrieval and transformations of structured documents. Structured documents are defined by grammars and stored as parse trees in files. The retrieval by filtering comprises indexing of documents, definition of filters, and application of a filter to a set of documents. The indexed documents are stored in files and depict hierarchic structures of documents. Filters are generated in a syntax-directed way according to the grammar. They define constraints for the structural elements of documents. Filters are stored in files and may be used in later search processes or in other filters. Before retrieval of documents, the output form and device may be specified.

The document management model of SYNDOC is suitable, for example, for SGML documents. When using the model, parsed documents with the same DTD can be collected in a directory and indexed separately. The user does not collect a large database or repository as, for example, in Dyntext [4], but can consider separate subsets of documents in the directory. The problem of updates which occurs in databases does not exist. Documents can be updated, deleted or inserted without effecting filtering of other documents.

In the following sections, this paper briefly describes the model for the definition of two-dimensional filters. The third and fourth sections present the architecture of SYNDOC and, phase by phase, the search for documents from the user's point of view. The final section comprises conclusion.

2 TWO-DIMENSIONAL FILTERS FOR STRUCTURED TEXTS

In our filtering approach, which we have defined theoretically in [18], a two-dimensional representation of a grammar is used as a template of a filter, and a filter is derived by adding constraints and annotations to the template. Two dimensions of the filter describe parent and sibling relations in the hierarchic text structure. In the following sections we present the main concepts of the model. The details can be found in [18].

2.1 Grammars as definitions of text types

A *context-free grammar* defines a set of terminal symbols, a set of nonterminal symbols to represent structural elements, a specific nonterminal symbol called the start symbol, and a set of productions to show how structural elements are composed of other elements. In the text model described in [19,20,21] and applied in [18], a context-free grammar is regarded as a definition for a set of text types, and a parse tree for the grammar as a hierarchy of instances called *parts* of the types. The names of the text types appear as nonterminal symbols in the grammar.

```
(1)  article        --> authors [date] title content
(2)  authors        --> author+
(3)  content        --> abstract section+
(4)  section        --> heading paragraph+
(5)  paragraph      --> |text_para |itemizelist
(6)  itemizelist    --> itemize+
```

Figure 1. Grammar productions for an article

Consider, for example, the productions in the Fig. 1. The metasymbols associated with nonterminals on the right side of productions indicate iteration (`*` for zero or more times and `+` for one or more times), alternatives (`|`) and optionality (`[` and `]`). The productions define the text types `article`, `authors`, `content`, `section`, `paragraph` and `itemizelist`. Productions (1) – (4) and (6) define types whose parts consist of one or more parts of other types. Production (5) defines parts of type `paragraph` being either parts of type `textpara` or parts of type `itemizelist`. The nonterminal symbols appearing on the right side of productions but not on the left side of any production are all regarded as text types whose parts consist of word sequences. For example, in the sample grammar `title` is a type defined by the implicit production

```
title --> word+
```

where type `word` represents a terminal symbol of the grammar accepted as a word constant in the language. A parse tree for the sample grammar is shown in the Fig. 2.

The parse tree is a containment hierarchy of the parts of the types defined by the sample grammar. The parts are represented in the tree by nodes labelled by text types. The root of the tree is a part of the type defined as the start symbol of the grammar. All other nodes labelled by types are parts except the nodes which have no siblings. For example, the nodes labelled by `paragraph` are parts while the nodes labelled by `textpara` are not; they only indicate that the parts of type `paragraph` are also parts of type `textpara`. The subtree with a part as its root is the *content* of the part and the string produced by concatenating the terminal symbols of the subtree (in their preorder) is the *value* of the part. A part *x contains* part *y* is *y* is a node in the content of *x*.

Each of the text types of a grammar can be considered as a logical operation which tests if a part of a parse tree is a part of the type. In addition to text types, also other logical operations may be defined for parts by associating a text type with a condition denoted by braces. The operations are called *properties*. In [18], the following properties were

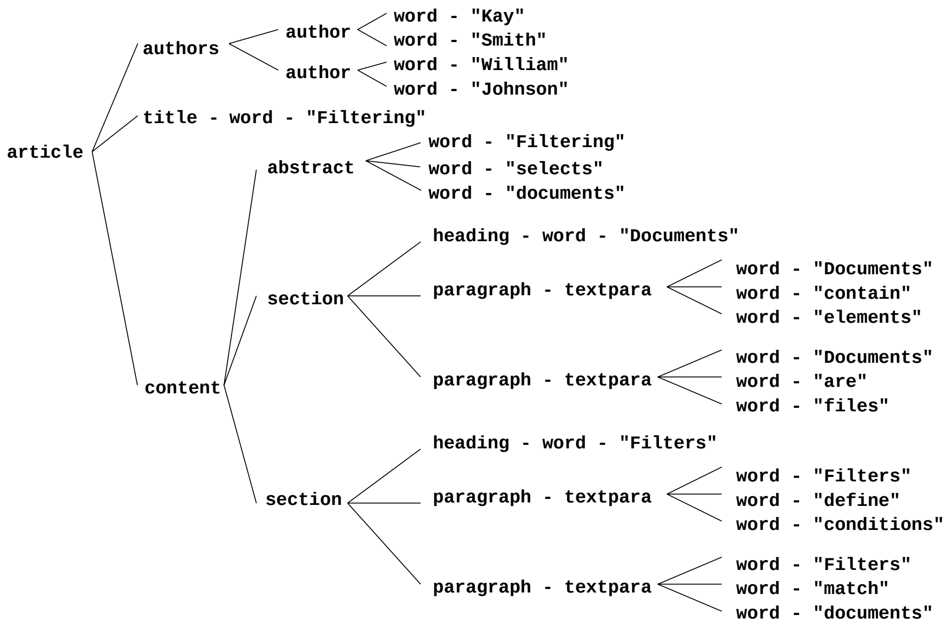


Figure 2. Parse for the grammar in Fig. 1

represented. In the following descriptions t and t_1 are text types, s is a character string, n_1 and n_2 are positive or negative integers and $q_1, q_2, \ldots, q_k$ are constraints. The text in the right column defines those cases in which the property is true for a part of the type t.

$t\{s\}$	the value contains string s as its substring
$t\{= t_1\}$	the value of the part equals the value of some part of type t_1
$t\{t_1\}$	contains a part of type t_1
$t\{n_1..n_2\}$	the position of the part among siblings of the same type is between the numbers n_1 and n_2
$t\{q_1 \,\&\, q_2 \,\&\, \ldots \,\&\, q_k\}$	all of the properties $t\{q_i\}$ are true
$t\{q_1 \mid q_2 \mid \ldots \mid q_k\}$	at least one of the properties $t\{q_i\}$ is true
$t\{!\, q_1\}$	the property $t\{q_1\}$ is not true

For example, the property `author`{`"Smith"`} is true for the `author` parts which contain the word Smith and the property `author`{`"Smith" & 1..2`} indicates that the word Smith has to be in the first or second `author` part.

2.2 Templates

A *template* is created for one of the text types of the grammar to depict the structure defined for the parts of the text type, at a chosen level of detail. Formally, a template is defined as an ordered labelled tree whose root is labelled by the type for which the template is generated.

A template as a tree is visualized such that each node label is written on its own line, and the parent-child relationship is expressed by indentation. The following are templates for an article generated from the previous grammar. The root of both of the templates is type `article`. The left-hand side template shows the main components of an article. The right-hand side shows also the structure of the `authors` component.

```
article                 article
    authors                 authors
    [date]                      author+
    title                   [date]
    content                 title
                            content
```

In [18], the correspondence between text type occurrences of a template and parts of a parse tree is defined. To set restrictions to parts corresponding to text type occurrences, a *constrained template* is formed by adding constraints defined for properties to type occurrences of the template. Constraints are written on the right side of a type occurrence. The following constrained filter

```
article
    authors
        author+........."smith" & 1..2
    [date]
    title
    content
```

defines articles having Smith as the first or second author. The semantics of a constrained template is specified in [18] by defining those cases where a part of a parse tree matches a constrained template.

For iterative text type occurrences, expressed by metasymbols `*` or `+` in templates, a *quantity constraint* may be added to express the number of parts for which the property associated with the text type must be true. The quantity constraint is one of the following: ALL, n, $> n$, $\geq n$, $< n$ or $\leq n$ where n is a non-negative integer.

2.3 Filters

A *filter* consists of constrained templates where the specified parts are indicated by *annotations*. While the text types are schema-level definitions, annotations allow query-level definitions. Names of annotations cannot be names of text types. Annotations are written on the left side of a type occurrence in a template.

A *simple filter* is a constrained template where one or more text type occurrences are annotated. A *compound filter* is a sequence of simple filters bound with the use of annotations. The annotations are regarded as definitions of *dynamic text types*. A dynamic type defined in a simple filter may be used in constraints of the subsequent filters. In [18] it is defined in which cases a part of a parse tree matches an annotation in a filter.

Let us consider the following compound filter.

```
long_chan_art....article
                    authors
                       author+..........."Chan"
                    [date]
                    title
                    content
                       abstract
                       section+..........QTY: >4


short_smith_art..article
                    authors
                       author+..........."Smith" & 1..2
                    [date]
                    title
                    content
                       abstract
                       section+..........QTY: =<4

art...............article................long_chan_art | short_smith_art
```

The annotation `art` in the last simple filter specifies articles which are either long (more than four sections) written by Chan, or short (less than or equal to four sections) written by Smith as the first or second author. The first condition is defined by the first simple filter, the second one by the second filter. The annotations of the first two simple filters define dynamic text types `long_chan_art` and `short_smith_art`, which are then used to specify the required article in the third filter.

More examples of the use of filters can be found in [18].

3 SYNDOC, A SYNTAX-DIRECTED DOCUMENT PROCESSING SYSTEM

Syntax-directed text processing in the SYNDOC environment [22,23] is based on the syntax-directed translation defined in the theory of formal languages [24]. SYNDOC uses context-free grammars and their parse trees in the input, update, retrieval, and output of documents. Figure 3 describes the architecture of the system depicting the processing phases of structured documents according to one grammar.

The structure grammar defines the type for a set of documents and is called a *skeleton grammar*. It contains only nonterminal symbols, among them a special nonterminal for the characters strings of the content. The creation of documents starts by defining a skeleton grammar. On the basis of the skeleton grammar, input grammars are defined for the input and output grammars for the output of documents. In an input grammar, the advice-information for the user is given using terminals interleaved with nonterminals of the skeleton grammar. In the output grammar, layout codes are those terminals which are added to the skeleton grammar. From input and output grammars the system compiles the input and output programs, respectively.

The input program is a syntax-directed editor which incrementally creates a parse tree according to the skeleton grammar from the pieces of text written by the user. Documents formatted for printing, viewing and for external representations are produced by the output programs called output generators.

For the document retrieval, a filter defining the conditions for selected documents is created from the skeleton grammar using a filter editor. The filtering engine compares the

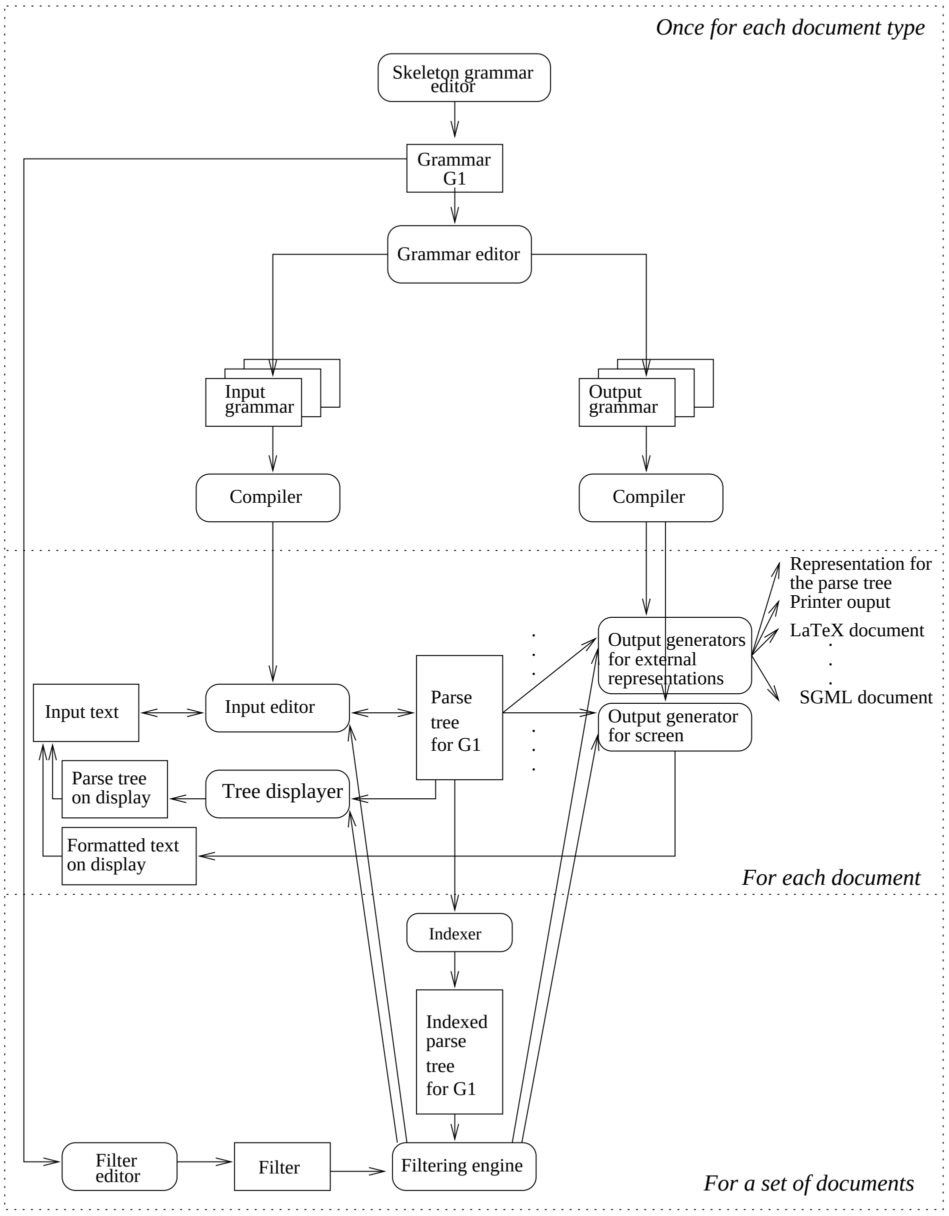


Figure 3. Architecture of SYNDOC

filter to indexed parse trees which an indexer has created from parse trees. The selected documents are then either edited using the input editor and viewed by the tree displayer, or displayed or printed using user-defined output generators.

SYNDOC has an accompanying grammar-based document transformation system called TRANSDOC [25,26] which makes parse tree transformations using various tree transformation methods. Transformation definitions for different methods are created using two skeleton grammars, either automatically or with user's help. Automatic transformations are made from one parse tree to another. In filtering it is possible that selected documents are first transformed according to a new structure grammar and then displayed or printed using the new structure.

4 RETRIEVAL BY FILTERING

The retrieval of documents in SYNDOC is based on the filtering approach and consists of three activities that can be executed individually:

1. indexing to allow the user to index documents,
2. filter definition to allow the user to specify constraints, and
3. filtering to allow the user to compare a filter with indexed documents and to display or print documents in a requested form.

Before the user can execute these activities, she or he must select a grammar which restricts the amount of source documents in filtering. Indexing and filtering are applied only to documents valid for the selected grammar from which the filter is derived. Conditions in filters that do not conform to the selected grammar are not allowed. The grammar is also used to inform the user of ambiguous constraints. We call this kind of approach a *syntax-directed filtering*.

Filters are long-term definitions and are not used only during a search process. Also indexed documents and retrieved documents may be saved for further purposes. All of them can be used by the same and other users.

In SYNDOC, the search starts from the main menu using the menu item **Retrieval** (Fig. 4). Other items of the main menu allow the user to create grammars as well as input, output and transform documents. **Retrieval** selection opens a window which contains a field to determine a skeleton grammar and buttons for different phases of the retrieval: **Indexing**, **Defining Filters**, and **Filtering**. The user cannot continue the filtering process without first defining the grammar. The aim is to compare the filter with all documents for the same skeleton grammar in the working directory.

4.1 Indexing documents

Indexing of documents for a grammar starts from menu item **Indexing** (Fig. 4). The user indexes selected files or all files for the selected grammar in the working directory. If the user indexes selected files, their names are provided via a document selection window which lists all document files for the grammar in the working directory. The system uses an index control file to direct indexing. User-defined control files may contain words which are not indexed, as well as other control information (for example, the method used for indexing). The index as an inverted list indicates occurrences of words and elements in the document and depicts the hierarchic structure of elements. For each document file a separate index file is created.

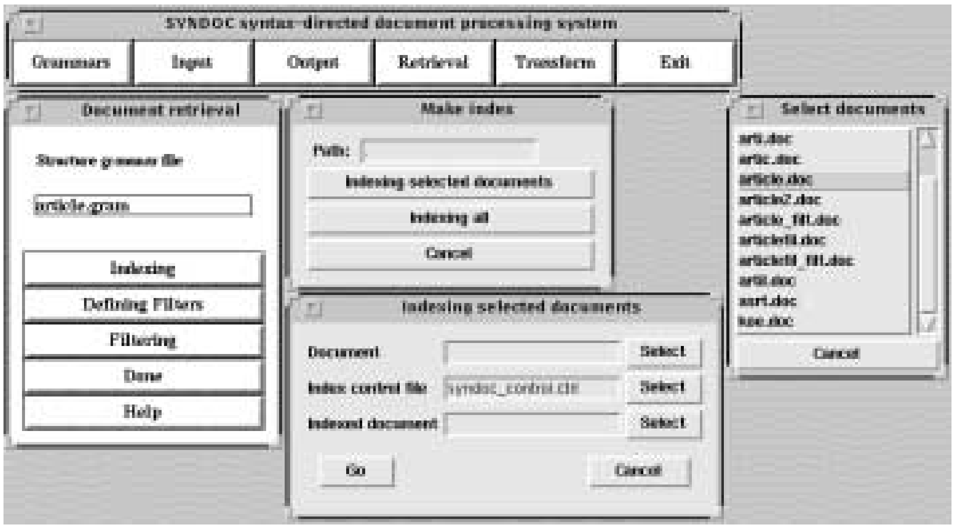


Figure 4. Selecting documents for indexing

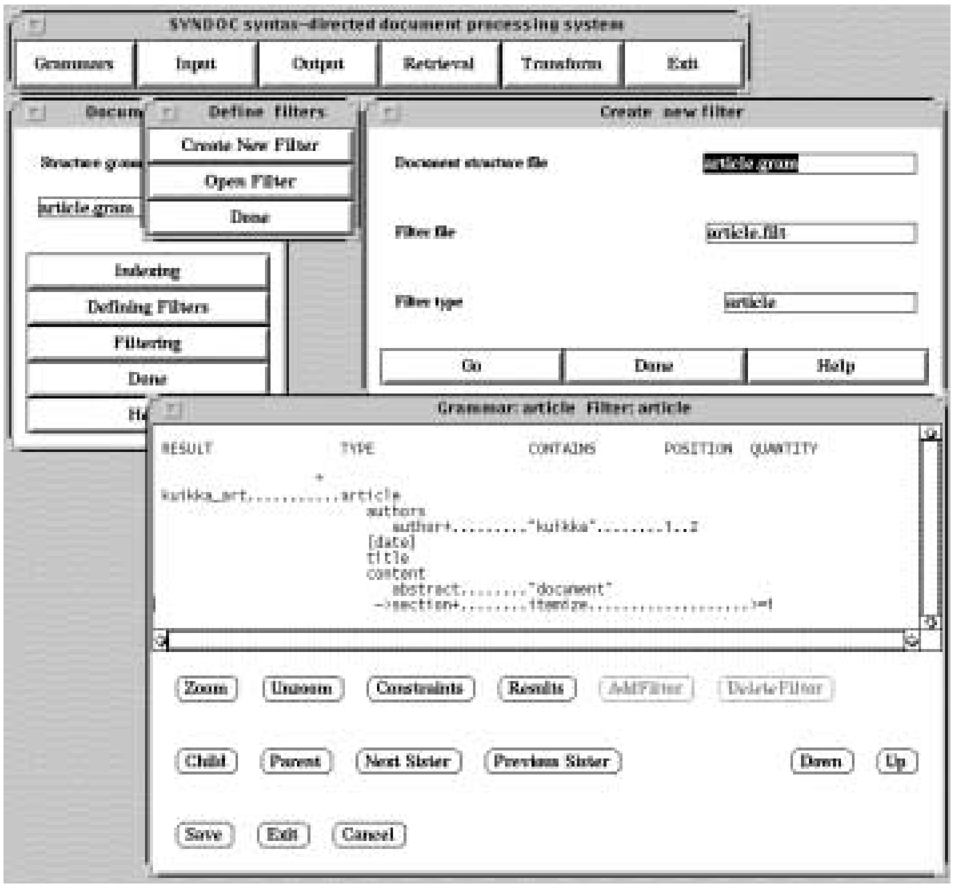
4.2 Defining filters

The user initiates the filter generation by giving file names for a grammar and filter, and the name of a text type for the root of the filter (Fig. 5). For an existing filter, only a name of the file containing the filter definition is selected.

On the screen, a filter is represented as a table with five columns (Fig. 5). The first column (`RESULT`) is for annotations. The second column (`TYPE`) shows a template expressing the hierarchical text structure. Columns three, four and five (`CONTAINS`, `POSITION` and `QUANTITY`) are for constraints. A containment constraint is a type, a type preceded by =, a character string, or a Boolean combination of them. Position constraints are of the form n_1 or $n_1..n_2$ where n_1 and n_2 are positive or negative integers. If both a containment constraint q_1 and a position constraint q_2 has been associated with a type t, it denotes the property $t\{q_1\&q_2\}$.

The template of a filter is generated from its root type in a syntax-directed way from the skeleton grammar. The user creates the template using the buttons **Zoom**, **Unzoom**, **Constraints**, **Results**, **AddFilter** and **DeleteFilter**. By zooming, the user adds type occurrences on the right side of the production for the current text type. The system does not allow illegal constraints. Also it is checked that a new annotation is distinct from the names of text types of the grammar and from previous annotations of the filter.

The buttons **Child**, **Parent**, **Next Sister**, **Previous Sister**, **Down** and **Up** allow the user to move from a type to another in the tree and expand and define constraints and annotations is any order. Buttons **Save**, **Exit** and **Cancel** are used in saving the created filter.



Structure gram

article.gram

Document structure file article.gram

Filter file article.filt

Filter type article

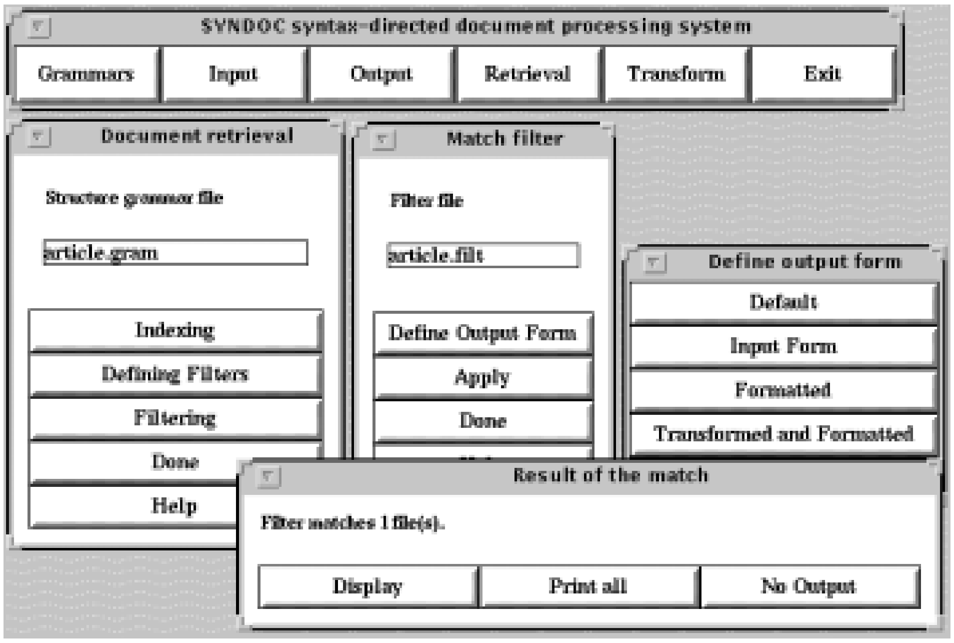
```
RESULT              TYPE                  CONTAINS        POSITION  QUANTITY

                 +
kuikka_art...........article
                       authors
                         author+.........."kuikka".........1..2
                       [date]
                       title
                       content
                         abstract........"document"
                        ->section+.........itemize.....................>=1
```

Figure 5. Filter on the screen

4.3 Filtering

Comparing a filter to indexed documents is initiated by using the **Filtering** menu item (Fig. 6). The user selects the filter by its file name. Search can be carried out among selected indexed documents or all indexed documents in the working directory. The comparison test counts the documents for which the filter matches and informs the user how many documents have been found (Fig. 6).

The user defines the output form for selected documents using the **Define Output Form** (Fig. 6). The output for selected documents can be generated in four different forms: the default form, input form, formatted form, and transformed and formatted form. The default form shows documents in the form in which they are stored. The input form opens the input editor and allows the user to edit the document. The layout form represents a formatted form using a previewer. We have produced a formatted layout as ASCII text containing line feeds between structure elements, and using Hypertext Markup Language (HTML) or



Structure grammar file: article.gram

Filter file: article.filt

Filter matches 1 file(s).

Figure 6. Displaying the result of searching on the screen

the LATEX formatter. The transformed and formatted form allows the user to modify the document structures and then format the transformed documents.

5 SYNDOC IMPLEMENTATION

SYNDOC has been implemented in SICStus Prolog (version 2.1 patch #9) [27] with the Graphics Manager library to create the X-window user interface, and in Tcl (version 7.4) and Tk toolkit (version 4.0) script languages [28]. The system runs in Sun workstations within the unix operating system.

The main aim of the implementation of the retrieval module has been the filter generation module and co-work with other modules. The indexing method is described by Burkowski in [29]. The indices indicate occurrences of indexed words and as well as occurrences of parts containing indexed words. Words of documents are numbered to identify their positions. Other parts are identified using the position numbers of their first and last words. The matching algorithm compares simple filters of a compound filter starting from the first one. For each simple filter, the algorithm searches all indexed parts of a document for the root text type of the filter and checks if the filter is true for one of them. The comparison of children of the text type *t* in the filter is restricted to parts inside the parts which correspond to type *t*. The algorithms and programs are described in detail in [23].

The current SYNDOC implementation is a prototype where documents are represented by Prolog terms. In the implementation only short documents have been tested. The transfer of the system to accept larger documents will be an area of our future research.

6 CONCLUSIONS

In this paper we have presented a method to retrieve document files using the filtering approach. The method is implemented in a syntax-directed document processing system which uses a syntax-directed translation as a processing model and allows documents to be represented as parse trees for a grammar, and created and formatted separately. In the document retrieval, indexing, filter definition, and filtering consisting of editing, browsing or viewing documents are made separately and not tightly connected to each other. The output of various activities may be stored to files for further purposes.

The filtering system allows the user to group document files with the same structure in the same dictionary (or many dictionaries if documents with the same structure need to be processed as subsets) and then retrieve those documents only. Filters offer a graphical way to define queries integrating structure and content conditions. Furthermore, the user can use existing filters either for a new retrieval or as part of a new filter. Finally, the user can define different kinds of layouts for selected documents for various purposes.

ACKNOWLEDGEMENTS

This work was financed by the Acadamy of Finland and supported by grants of the Saastamoinen Foundation, the Emil Aaltonen Foundation, and Jenny and Antti Wihuri Foundation, which are gratefully acknowledged. The authors would like to thank Martti Penttonen, Jorma Sajaniemi, Pekka Savolainen, Heikki Mannila, Pekka Kilpeläinen, Erja Nikunen, Helena Ahonen and Frank Tompa for their comments and Jouni Mykkänen and Arto Ryynänen for the programming work.

REFERENCES

1. N.J. Belkin and W.B. Croft, ‘Information filtering and retrieval: two sides of the same coin?’, *Communications of the ACM*, **35**, 29–38, (1992).
2. J. Jaakkola and P. Kilpeläinen, ‘Sgrep - a tool to search structured text’, Technical report, Department of Computer Science, University of Helsinki, Finland, (1995). In preparation.
3. C.L.A. Clarke and G.V. Cormack, ‘The use of regular expression for searching text’, Technical Report CS-95-07, University of Waterloo, Department of Computer Science, (1995).
4. EBT, *DynaText Publisher Guide I & II*, 1993.
5. I.A. Macleod, ‘Storage and retrieval of structured documents’, *Information Processing & Management*, **26**(2), 197–208, (1990).
6. I.A. Macleod, B.T. Barnard, Hamilton D., and M. Levison, ‘SGML documents and non-linear text retrieval’, in *RIAO’91, Proceedings of the Conference on Intelligent Text and Image Handling*, pp. 226–244, (1991).
7. E. Bertino, F. Rabitti, and S. Gibbs, ‘Query processing in a multimedia document system’, *ACM Transactions on Office Information Systems*, **6**(1), 1–41, (1988).
8. S. Abiteboul, S. Cluet, and T. Milo, ‘Quering and updating the file’, in *Proceedings of the 19th VLDB Conference*, pp. 73–84, Dublin, Ireland, (1993).
9. S. Abiteboul, S. Cluet, and T. Milo, ‘A database interface for files update’, in *SIGMOD Record*, San Jose, California, (1995).
10. M.P. Consens and T. Milo, ‘Optimizing queries on files’, in *SIGMOD’94, Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, eds., R.T. Snodgrass and M. Winslett, pp. 301–312, (1994). SIGMOD Record, Vol. 23, Issue 2, June 1994.
11. M.P. Consens and T. Milo, ‘Algebras for querying text regions’, in *Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, PODS 1995*, pp. 11–22, San Jose, California, (1995).

12. A. Loeffen, ‘Text databases: A survey of text models and systems’, *SIGMOD RECORD*, **23**(1), 97–106, (1994).
13. R. Sacks-Davis, T. Arnold-Moore, and J. Zobel, ‘Database systems for structured documents’, in *International Symposium on Advanced Database Technologies and Their Integration (ADTI’94)*, pp. 272–283, Nara, Japan, (1994).
14. R. Baeza-Yates and G. Navarro, ‘Integrating contents and structure in text retrieval’, *SIGMOD Record*, **25**(1), 67–79, (1996).
15. M.M. Zloof, ‘Query-by-example: a database language’, *IBM System Journal*, **16**(4), 324–343, (1977).
16. G. Özsoyoglu and H. Wang, ‘Example-based graphical database query languages’, *Computer*, **May**, 25–38, (1993).
17. M.S. Neff, R.J. Byrd, and O.A. Rizk, ‘Creating and quering lexical data bases’, in *Proceedings of the 2nd Conference on Applied Natural Language Processing ACL*, pp. 84–92, (1988).
18. E. Kuikka and A. Salminen, ‘Two-dimensional filters for structured text’. To appear in *Information Processing & Management*.
19. A. Salminen and C. Watters, ‘Two-level structure for textual databases to support hypertext access’, *Journal of the American Society for Information Science*, **43**(6), 432–447, (1992).
20. A. Salminen, J. Tague-Sutcliffe, and C. McClellan, ‘From text to hypertext by indexing’, *ACM Transactions on Information Systems*, **13**(1), 69–99, (1995).
21. A. Salminen and F.W. Tompa, ‘Grammars++ for modelling information in text’, Technical report, UW Centre for the New OED and Text Research, University of Waterloo, Department of Computer Science, (1995). In preparation.
22. E. Kuikka, M. Penttonen, and M.-K. Väisänen, ‘Theory and implementation of SYNDOC document processing system’, in *Proceedings of the Second International Conference on Practical Application of Prolog*, pp. 311–327, London, UK, (April 1994).
23. E. Kuikka, J. Mykkänen, A. Ryynänen, and A. Salminen, ‘Implementation of two-dimensional filters for structured documents in SYNDOC environment’, Technical Report A-1995-4, Department of Computer Science, University of Joensuu, (1995).
24. A.V. Aho and J.D. Ullman, *The theory of parsing, translation, and compiling, Vol. I: Parsing*, Prentice-Hall, Inc., Englewood Cliffs, N.J., USA, 1972.
25. E. Kuikka and M. Penttonen, ‘Transformation of structured documents with the use of grammar’, *Electronic Publishing - Origination, Dissemination and Design*, **6**(4), 373–383, (1993).
26. E. Kuikka and M. Penttonen, ‘Transformation of structured documents’. Submitted for publication.
27. Swedish Institute of Computer Science, *SICStus Prolog User manual and SICStus Prolog Library manual (Version 2.1)*, 1993.
28. J.K. Ousterhout, ‘Tcl and the tk toolkit’. Draft version of ISBN 0-201-63337-X, 1993.
29. F.J. Burkowski, ‘An algebra for hierarchically organized text-dominated databases’, *Information Processing & Management*, **28**(3), 333–348, (1992).