Copyright Warning & Restrictions

The copyright law of the United States (Title 17, United States Code) governs the making of photocopies or other reproductions of copyrighted material.

Under certain conditions specified in the law, libraries and archives are authorized to furnish a photocopy or other reproduction. One of these specified conditions is that the photocopy or reproduction is not to be "used for any purpose other than private study, scholarship, or research." If a, user makes a request for, or later uses, a photocopy or reproduction for purposes in excess of "fair use" that user may be liable for copyright infringement,

This institution reserves the right to refuse to accept a copying order if, in its judgment, fulfillment of the order would involve violation of copyright law.

Please Note: The author retains the copyright while the New Jersey Institute of Technology reserves the right to distribute this thesis or dissertation

Printing note: If you do not wish to print this page, then select "Pages from: first page # to: last page #" on the print dialog screen

ABSTRACT

A GRAPHICAL ENVIRONMENT FOR CHANGE DETECTION IN STRUCTURED DOCUMENTS

by
Girish A. Patel

Change detection in structured documents (e.g. SGML) is important in data warehousing, digital libraries and Internet databases. This thesis presents a graphical environment for detecting changes in the structured documents. We represent each document by an ordered labeled tree based on the underlying markup language. We then compare two documents by invoking previously developed algorithms for approximate pattern matching and pattern discovery in trees. Several operators are developed to support the comparison of the documents; graphical devices are provided to facilitate the use of the operators. We believe the proposed tool is useful for not only document management, but also software maintenance, particularly configuration management and version control, where programs are represented as parse trees and detecting changes in the trees provides a way to find the syntactic differences of two program versions.

A GRAPHICAL ENVIRONMENT FOR CHANGE DETECTION IN STRUCTURED DOCUMENTS

by
Girish A. Patel

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Computer Science

Department of Computer and Information Science

May 1997

APPROVAL PAGE

A GRAPHICAL ENVIRONMENT FOR
CHANGE DETECTION IN STRUCTURED DOCUMENTS

Girish A. Patel

Dr. Jason T. L. Wang, Thesis Advisor — Date
Associate Professor of Computer and
Information Science Department,
New Jersey Institute of Technology

Dr. James McHugh, Committee Member — Date
Professor of Computer and Information Science Department,
New Jersey Institute of Technology

Dr. Peter A. Ng, Committee Member — Date
Chairman and Professor of Computer and
Information Science Department,
New Jersey Institute of Technology

BIOGRAPHICAL SKETCH

Author: Girish A. Patel

Degree: Master of Science in Computer Science

Undergraduate and Graduate Education:

- Master of Science in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 1997
- Bachelor of Science in Electrical Engineering,
  Birla Vishvakarma Mahavidyalaya, India, 1988

Major: Computer Science

Presentations and Publications:

J. T. L. Wang, D. Shasha, G. J. S. Chang, G. A. Patel, L. Relihan and K. Zhang. NJIT, NYU, Piercom Ltd., University of Western Ontario. *Structural Matching and Discovery in Document Databases.* To appear in ACM SIGMOD, May 1997.

J. T. L. Wang, G. J. S. Chang, L. Relihan and G. A. Patel. *A Graphical Environment for Change Detection in Structured Documents.* To appear in COMPSAC97, Washington D.C., August 1997.

To
my parents
my wife Daxa Patel and
my daughter Tithi Patel

ACKNOWLEDGMENT

I wish to thank my advisor, Dr. Jason T. L. Wang, for his guidance, encouragement and support throughout this thesis.

Special thanks to George J. S. Chang who helps me in getting started. I spent many hours with him discussing various approaches and ideas, for this I am sincerely grateful.

I also wish to thank Dr. James McHugh and Dr. Peter A. Ng for serving as members of the committee.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

It has recently been the trend in document systems technology to emphasize the logical structures inherent in many kinds of documents. In general, text processing systems and word processing systems require additional information to be recorded on the text of the document being processed. This metainformation is usually interspersed among the actual text itself and is often referred to as markup. Individual fragments of markup are called tags.

One kind of markup - generalized markup - is becoming increasingly common. Generalized Markup is based on two postulates [7, 9]:

- Markup should describe a document's structure and other attributes rather than specifying processing instructions to be carried out on it.
- Markup should be rigorous in order that techniques available for processing other rigorously defined objects (e.g. programs, databases) be available for processing documents also [1].

Generalized markup provides the following advantages over the more usual kind of markup (descriptive markup) that merely specifies processing instructions:

- Information is preserved; the identification of logical elements is not lost.
- Arbitrary processing instructions may be assigned to tags. This provides:
  - flexibility: the appearance of whole sets of documents may be changed instantly by simply changing the processing instructions associated with the tags;

  - portability: since platform dependent processing instructions are not embedded in descriptively marked-up documents, the portability of documents is enhanced.

- It is feasible to make "intelligent" queries on documents [4].

SGML is a metasyntax that is used for writing generalized markup syntaxes. Ultimately, the rationale behind SGML is to provide mechanisms that allow documents to be described in such a way that they are easily portable across systems. In fact, the SGML language is becoming *de facto* the standard for structured document creation and exchange.

This thesis presents a graphical environment for change detection in structured documents such as SGML and its extension HTML. The SGML and HTML are widely used to define document types for the defense [10], aerospace, publishing industries and World Wide Web. Detecting changes to the structured documents is a basic function of many important applications, including data warehousing, digital libraries, version management, hypermedia and Internet databases [3, 8, 11, 12, 21]. As an example, a user of the World Wide Web may be interested in knowing changes in an HTML document. Such changes can be detected by comparing the old and new version of the document. As another example, in hypertext authoring, a user may wish to find the common portions in the history list of a document or a set of documents.

Our approach to detecting document changes is first to translate a document into an ordered labeled tree structure based on the underlying markup language. (An ordered labeled tree is a tree in which each node has a label and the left to right order of its children, if it has any, is fixed.) We then compare two documents (trees) using approximate tree matching techniques which find a minimum number of tree edit operations (insert node, delete node and relabel node) for transforming one tree

to the other. Besides, our system can find an approximately common portions of a set of documents by invoking a previously developed algorithm for pattern discovery in trees. We believe that the proposed techniques should help not only document management, but also software maintenance, particularly configuration management and version control [13], where programs are represented by parse trees and changes to the code can be detected by comparing the parse trees.

The rest of the thesis is organized as follows. In Chapter 2 we review the background for the SGML language, the representation of structured documents using trees, and tree edit operations. In Chapter 3, we describe the architecture of our tool and the underlying algorithms. Chapter 4 presents the graphical user interface of the tool. Chapter 5 concludes the thesis.

CHAPTER 2

BACKGROUND

2.1 SGML Preliminaries

There are two important concepts in SGML: elements and entities. Elements are logical information components which compose a document. Common examples are sections, lists, paragraphs, etc. An SGML document consists of a hierarchical structure of elements called the element structure. Elements are of a defined type and can have a set of attributes. Element attributes are essentially pieces of information about an element.

SGML entities are units of information that may be referred to by a logical name in an SGML document. Entities are often used to hold strings of characters. They are also used to refer to typographical symbols that cannot be entered on ordinary computer keyboards and separate files that may or may not contain SGML data.

Arbitrary arrangements of elements and entities need not be permitted in SGML documents, since SGML provides for sets of rules that define the allowable contents of elements (content models). These sets of rules are contained in Document Type Definitions (DTDs) and all SGML documents must conform to a particular DTD. A document conforming to a DTD essentially consists of the document's content interspersed with tags that delimit elements within the content. For instance, Fig. 2.1 shows a DTD for a document of type article. Element names are used as tags in the document. The specification of an element in the DTD gives its name, its structure and some indications (e.g. "-O" indicates that the tag can be omitted if there is no ambiguity). The element structure is built using other elements or basic types such as #PCDATA, EMPTY, etc. and connectors that can be further qualified with occurrence indicators. In particular, the following can be used:

- The aggregation connector (",") implies an order between elements. For example, a figure is composed of a picture followed by a caption (line 10). There is also an alternative aggregation connector ("&") that does not imply an order.

- The choice connector ("—") provides an alternative in the type definition. For instance, element body is either a figure or a paragraph (line 9).

- The optional indicator ("?") indicates zero or one occurrence of an element (e.g., captions in figures (line 10)); the plus sign ("+") indicates one or more occurrences of an element (e.g., sections in articles, line 2); and the asterisk ("*") indicates zero or more occurrences of an element (line 7).

```
1.  <!DOCTYPE   article[
2.  <!ELEMENT   article     - -     (title,author+,affil,abstract,
                                    section+,acknowl)>
3.  <!ATTLIST   article             status (final|draft) draft
                                    #REQUIRED>
4.  <!ELEMENT   title       - -     (#PCDATA)>
5.  <!ELEMENT   author      - O     (#PCDATA)>
6.  <!ELEMENT   abstract    - O     (#PCDATA)>
7.  <!ELEMENT   section     - O     ((title,body+)|(title,body*,
                                    subsectn+))>
8.  <!ELEMENT   subsectn    - O     (title,body+)>
9.  <!ELEMENT   body        - O     (figure|paragr)>
10. <!ELEMENT   figure      - O     (picture, caption?)>
11. <!ELEMENT   picture     - O     EMPTY>
12. <!ATTLIST   picture             sizex NMTOKEN "16cm"
                                    sizey NMTOKEN #IMPLIED
                                    file ENTITY #IMPLIED>
13. <!ELEMENT   caption     O O     (#PCDATA)>
14. <!ENTITY    file        SYSTEM  "/u/george/TEX/SGML/fig1.ps"
                                    NDATA >
15. <!ELEMENT   paragr      - O     (#PCDATA)>
16. <!ELEMENT   acknowl     - O     (#PCDATA)> ]>
```

Figure 2.1 A DTD for a document of type article.

```
<article status="final">
<title> A Graphical Environment for Change Detection in
Structured Documents
<author> George J.S. Chang
<author> Girish Patel
<author> Liam Relihan
<author> Jason T.L. Wang
...
<abstract> Change detection in structured documents such as SGML
and HTML is important in many applications including data warehousing ...
<section>
<title> Introduction <\title>
...
<body><paragr> The rest of the thesis is organized as follows.
In Chapter 2 we review the background for the SGML language,
the representation of structured documents using trees, and tree edit
operations. In Chapter 3, we describe the graphical interface ...
<\body><\section>
<section>
<title> Background <\title>
<subsectn>
<title> SGML Preliminaries <\title>
<body><paragr> There are two important concepts in SGML: elements
and entities. Elements are logical information components ...
<\body><\subsectn><\section>
...
<\article>
```

Figure 2.2 An SGML document of type article.

Fig. 2.2 shows a document instance conforming to the DTD in Fig. 2.1 that contains the information content as well as the tags. This document instance can be translated to a tree structure as shown in Fig. 2.3. Thus comparing two documents amounts to comparing their tree structures.

2.2 Tree Comparison

We compare two documents (trees) by finding their edit distance. There are three types of edit operations, namely i.e., *relabeling*, *delete*, and *insert* a node. Relabeling

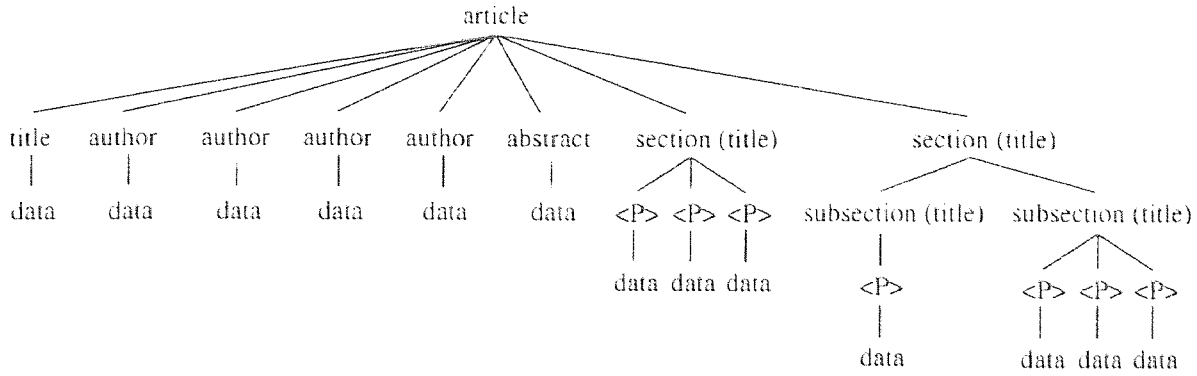

Figure 2.3 Tree representation of the document in Fig: 2.2.

node n means changing the label on n. Deleting a node n means making the children of n become the children of the parent of n and removing n. Insert is the inverse of delete. Inserting node n as the child of node n' makes n the parent of a consecutive subsequence of the current children of n'. Fig. 2.4 illustrate the edit operations.

Let S be a sequence $s_1, s_2, \ldots s_k$ of edit operations. S transforms tree T to tree T' if there is a sequence of trees $T_0, T_1, \ldots, T_k$ such that $T = T_0$, $T' = T_k$ and T_i results from T_{i-1} via s_i for $1 \leq i \leq k$. For the purpose of this work, we assume that all edit operations have unit cost. By extension, the cost of the sequence S, denoted $\gamma(S)$, is simply the sum of the costs of the constituent edit operations in S. The edit distance, or simply the *distance* from tree T_1 to tree T_2, denoted $dist(T_1, T_2)$, is the cost of a minimum cost sequence of edit operations transforming T_1 to T_2 [20].

The edit operations give rise to a mapping that is a graphical specification of which edit operations apply to each node in the two trees. For example, the mapping in Fig. 2.5 shows a way to transform T_1 to T_2. The transformation includes deleting the node labeled b in T_1 and inserting it into T_2. The cost of a mapping M is the cost of deleting nodes of T_1 not touched by a mapping line of M plus the cost of inserting nodes of T_2 not touched by a mapping line of M plus the cost of relabeling

the nodes related by a mapping line of M with different labels. It can be proved that the distance between two trees T_1 and T_2 equals the cost of a minimum cost mapping from T_1 to T_2 [22].

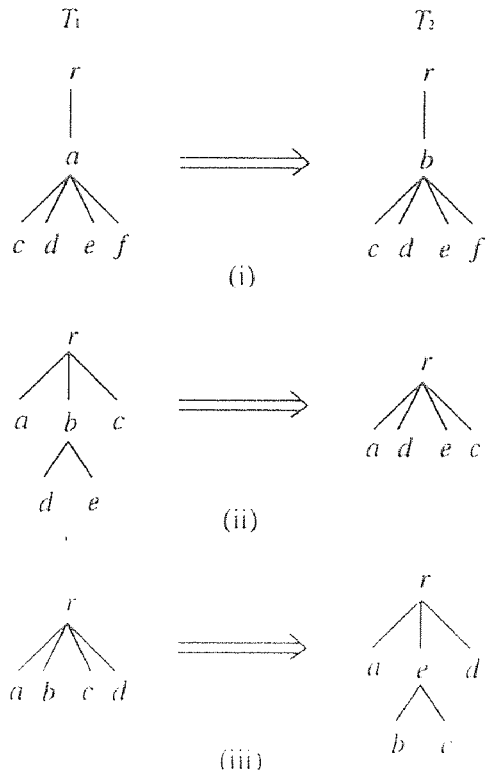

Figure 2.4 Examples illustrate the edit operations. (i) Relabeling: To change one node label (a) to another (b). (ii) Delete: To delete a node; all children of the deleted node (labeled b) become children of the parent (labeled r). (iii) Insert: To insert a node; a consecutive sequence of siblings among the children of the node labeled r (here, b and c) become the children of the node labeled e.

Given two trees (documents) T_1 and T_2 and an integer d, our system can also find the largest approximately common substructures, within distance d, of T_1 and T_2. A substructure of a tree T is a subtree of T with certain nodes being cut. (Cutting at a node n in T means removing the subtree rooted at n.) The largest approximately

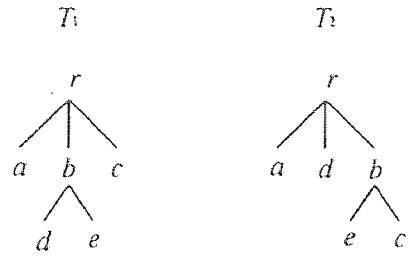

Figure 2.5 Example shows a mapping from tree T_1 to tree T_2. A dotted line from a node u in T_1 to a node v in T_2 indicates that u should be changed to (i.e. relabeled to) v if $u \neq v$, or that u remains unchanged if $u = v$. The nodes of T_1 not touched by a dotted line are to be deleted and the nodes of T_2 not touched by a dotted line are to be inserted. The mapping shows a way to transform T_1 to T_2.

common substructures, within distance d, of T_1 and T_2 refer to a substructure U_1 of T_1 and a substructure U_2 of T_2 such that U_1 is within distance d of U_2 and there does not exist any other substructure V_1 of T_1 and V_2 of T_2 such that V_1 and V_2 satisfy the distance constraint and the sum of the sizes of V_1 and V_2 is greater than the sum of the sizes of U_1 and U_2. When $d = 0$, U_1 (or U_2) is the largest common substructure of the two trees.

CHAPTER 3

ALGORITHMS AND ARCHITECTURE

3.1 Underlying Algorithms

Referring to Fig. 2.3, we represent the paragraphs of a document as leaves (terminal nodes) in the corresponding tree structure. The contents of the paragraphs are encoded into signatures [5]. Each signature is an integer value obtained by hashing the content of the corresponding paragraph. Thus, if two paragraphs are the same, their signatures remain the same. When the hash function chosen is perfect [6], two equivalent signatures also imply the equivalence of the corresponding paragraphs.

Likewise, the section and subsection titles associated with the nonterminal nodes of the tree structure are encoded into signatures. These signatures become the labels of the nodes. When comparing two documents (trees) T_1 and T_2, we use the approximate tree matching algorithm to find the best mapping that transforms T_1 to T_2. The algorithm runs in time $O(n_1 n_2(\min\{h_1, l_1\})(\min\{h_2, l_2\}))$ where n_1 (n_2, respectively) is the number of nodes, h_1 (h_2, respectively) is the height, and l_1 (l_2, respectively) is the number of leaves of T_1 (T_2, respectively) [20].

Our system can also detect the movement of paragraphs (i.e. moving one paragraph from one place to another) in T_1 and T_2. To do so, the system first finds the best mapping from T_1 to T_2. For leaf nodes not touched by mapping lines, compare their signatures. If the signature of a paragraph (leaf) in T_1 is found to be the same as the signature of a paragraph (leaf) in T_2, the two paragraphs should be the same and therefore a "move" message is displayed.

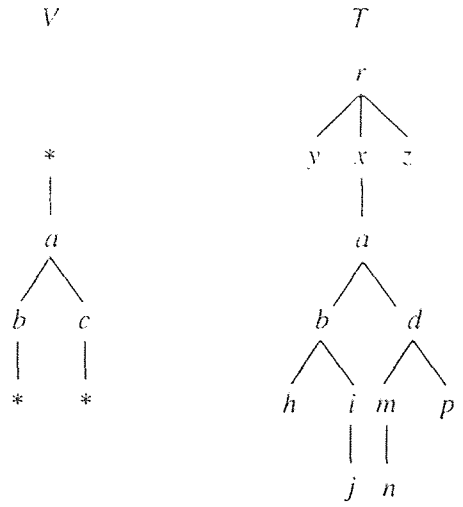

Figure 3.1 Example illustrate matching a VLDC pattern V and a tree T. The root $*$ in V would be matched with nodes r, x in T, and the two leaves $*$ in V would be matched with nodes i, j and m, n in T, respectively. Nodes y, z, h, p in T would be cut. The distance of V and T would be 1 (representing the cost of changing c in V to d in T).

In finding the largest approximately common portions (substructures), within a given distance value d, of T_1 and T_2, our algorithms run in time $O(dn_1n_2(\min\{h_1, l_1\})(\min\{h_2, l_2\}))$. By extension, our system can find the largest common portion of a set of documents. For example, in finding the largest common portion of three documents T_1, T_2, T_3, we first find the largest common portion P of two documents, say T_1 and T_2. Then compare P with T_3 to find their largest common portion.

To locate where a substructure M approximately occurs in a document T, we add variable length don't cares (VLDCs) to M as the new root and leaves to form a VLDC pattern V and then compare V with T using the pattern matching algorithm developed in [23]. (A VLDC (conventionally denoted by "$*$") can be matched, at no cost, with a path or portion of a path in T. This technique calculates the minimum distance between V and T after implicitly computing an optimal substitution for the

VLDCs in V, allowing zero or more cuttings at nodes from T (see Fig. 3.1).) The algorithm requires, in the worst case, $O(n_1 n_2(\min\{h_1, l_1\})(\min\{h_2, l_2\}))$.

3.2 System Architecture

Fig. 3.2 illustrate the architecture of the system. The back end of the system contains the programs for comparing trees whose algorithms are described in the previous subsection.

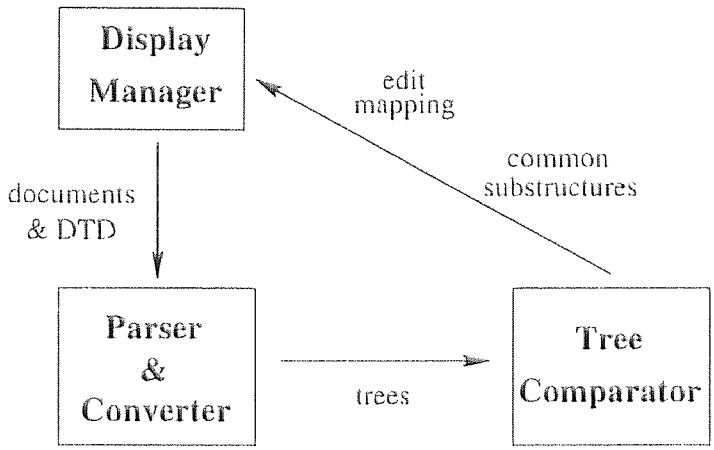

Figure 3.2 System architecture of the proposed graphical environment.

The front end of the system is composed of a display manager (to be described in the next section), a parser and a converter. The parser, written in the C++ programming language by James Clark, takes a DTD and a document as the input and parses the document according to the rules of the DTD. If a parsing error occurs (i.e., the document does not conform to the DTD), the parser rejects the document with an error message. On the other hand, if the document is parsed correctly, the parser produces a complete abstract syntax tree described in a format called ESIS (Element Structure Information Set). The converter, written in the interpreted Perl programming language, takes the ESIS description of the document's abstract syntax

tree and generates the output suitable for the back end of the system. The output consists of the following items:

- signatures used as node labels in the tree;
- the start and end line numbers of elements in the document, which facilitate the location and display of relevant textual objects by the back end;
- document component type identifiers for each element.

The converter does not attempt to impose any semantics upon the document being processed - it just treats the document as a tree structure whose nodes possess some associated information. It should be noted that our system is capable of working with many different document types (e.g. memos, letters, books, articles, etc.) - in fact any type describable with SGML. This use of a "generator-generator" approach allows us to remain independent of the semantics attached to any document types. Thus, given any two documents (or a set of documents), our system can compare them provided that they conform to the same DTD.

CHAPTER 4

GRAPHICAL INTERFACE

4.1 Graphical Interface

The display manager of our system is responsible for managing the graphical interface of the system. The system provides the user with a set of operators to select a DTD and documents conforming to the DTD. It then compares the documents, and displays/highlights the differences or the common portions of the documents through the graphical interface.

Fig. 4.1 shows the result of comparing two SGML documents. The two windows on the left show the documents. The right window displays the output of comparison. For each document element, only the first 8 characters are displayed. " =" indicates that the two document elements (e.g. two section titles or two paragraphs) are the same. ">" indicates that the corresponding document element is inserted, and "<" indicates the document element is deleted. The system can also show the movement of paragraphs (line ??). When the user clicks on the particular document element of interest in the right window, that portion of the document is scrolled to the top in its window. Fig. 4.2 shows the result of finding the largest common portion of three versions of a document. The places in which the common portion appears are highlighted by displaying them at the top of each document window.

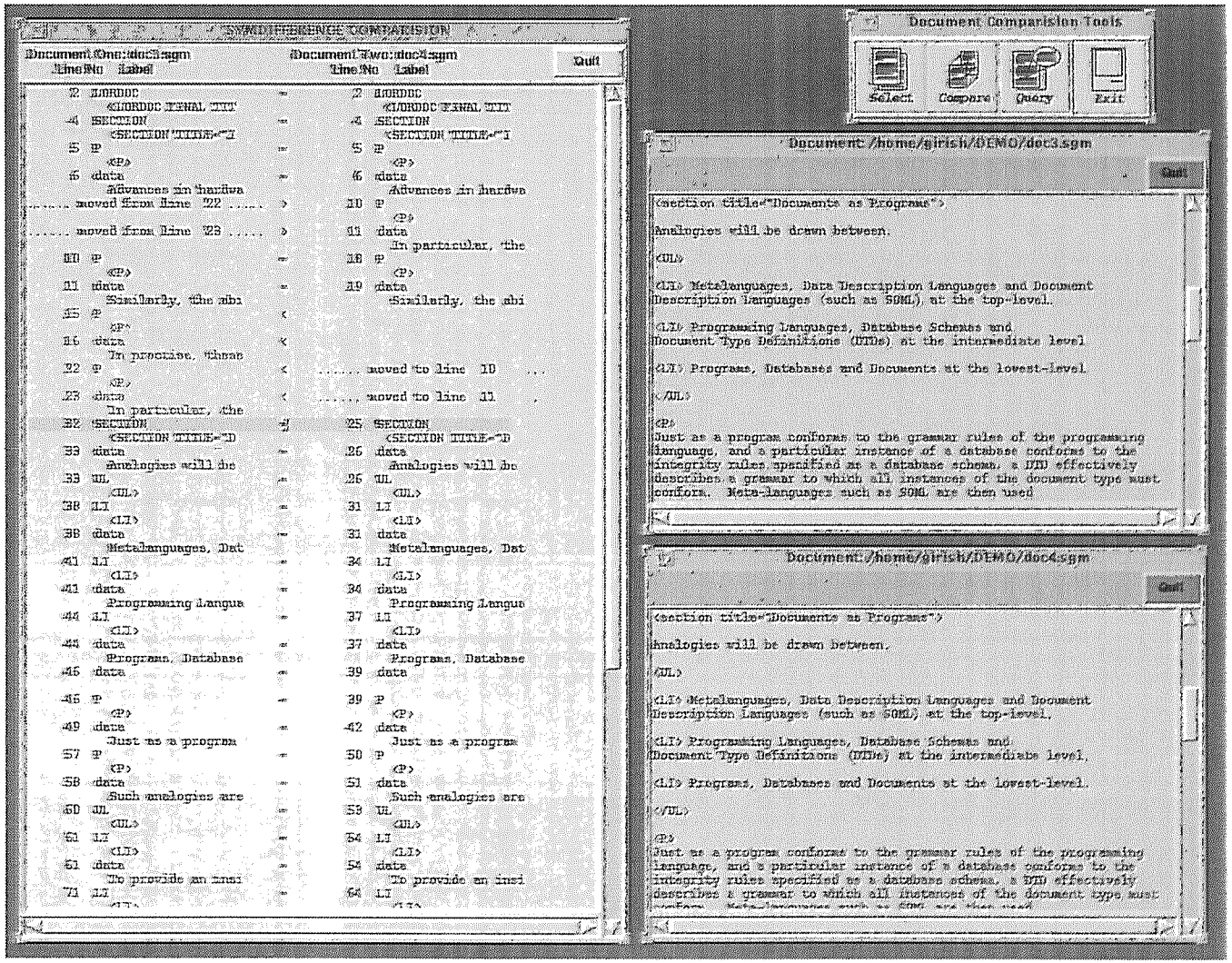

Figure 4.1 Result of comparing two SGML documents.

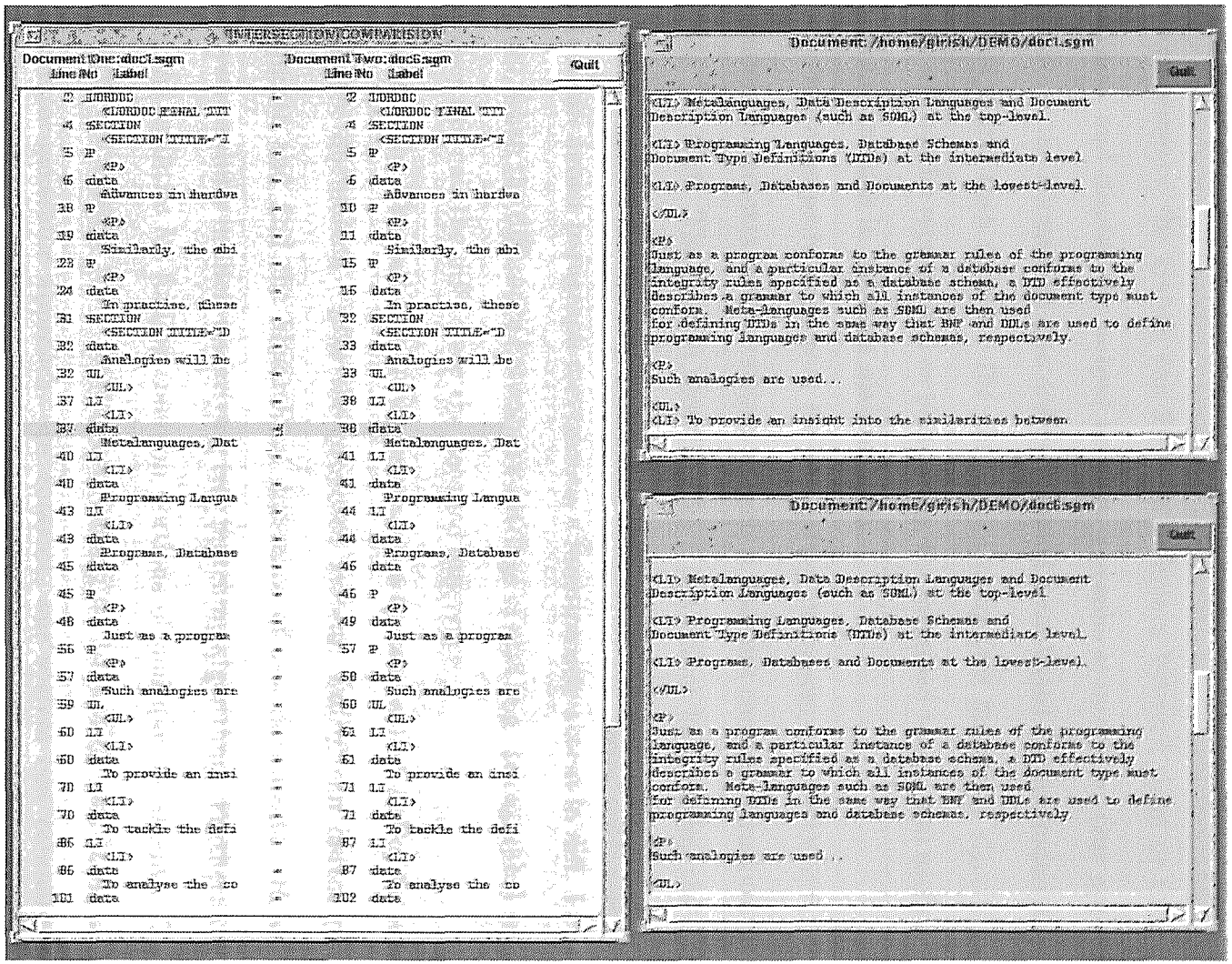

Figure 4.2 Result of finding the largest common portion of three versions of a document.

CHAPTER 5
CONCLUSION

This thesis presents a system for change detection in structured documents. Given two documents, the system can find the difference between them or their common portions, and display the output through a graphical interface. Our approach is to transform the documents into ordered labeled trees based on the underlying markup language and then compare the documents using tree matching algorithms. Since programs can be represented as parse trees, the proposed tool is also useful in tracing program versions in software maintenance.

One commonly used tool for comparing documents is UNIX diff. The tool considers documents as mere strings of text; it does not consider structure or markup within the text. The system described here can detect and display the differences in structured documents with respect to their hierarchical structure. Moreover, the system can detect paragraph movements, a functionality absent in diff. In using this system, when the difference between two strings of text (e.g. paragraphs) is detected, the user can call the diff to see the detailed difference.

Recently, Chawathe *et al.* [3] developed a system, called *LaDiff* for comparing two Latex documents. Like the proposed system, *LaDiff* can detect paragraph movements and find the hierarchical difference of two documents. However, the two systems differ in their underlying tree matching algorithms. Furthermore, *LaDiff* lacks the facilities for finding common portions of multiple documents. The types of target documents (i.e. Latex) are hardwired to *LaDiff* and it does not use a generator-generator approach. In contrast, our system can compare any two documents conforming to the same DTD and can deal with any SGML document type.

This system is implemented in C++, Perl, Tcl/Tk and run on a SunSpare workstation 20 under the Solaris operating system version 2.4. Currently, we are

incorporating the system into a visualization toolbox for pattern matching and discovery in scientific [2, 15, 16, 17, 18, 20] and document databases [14, 19]. The executable software packages of the toolbox are available from the authors.

APPENDIX A

PROGRAMS

This appendix contains all of the programs used in the implementation of graphical display module of tool.

```
 ------------------------------------------------------------------------
|                                                                        |
| Program: XSUN                                                          |
| Author:  Girish A.  Patel                                              |
|                                                                        |
| This program provides Graphical User Interface to the back-end         |
| program designed for change detection in structured documents          |
|________________________________________________________________________|

#!/local/bin/wish -f

global selectedlist
global pwdir
global OkFlag
global filetype
global OP
global level
set OP 0
set level 1
set OkFlag 0
set pwdir [pwd]
set algo_type "algo1"

wm title . "Document Comparison Tool"
frame .mbar -relief raised -bd 2
pack .mbar  -side top -fill x
button .mbar.exit  -bitmap @./bitmaps/exit.xbm -relief groove -command
exit pack .mbar.exit -side right -anchor n
menubutton .mbar.file -bitmap @./bitmaps/select.xbm -relief groove \
-menu .mbar.file.menu
menubutton .mbar.algo -bitmap @./bitmaps/compare.xbm -relief groove \
-menu .mbar.algo.menu
menubutton .mbar.search -bitmap @./bitmaps/search.xbm -relief groove \
-menu .mbar.search.menu
button .mbar.query -bitmap @./bitmaps/query.xbm -relief groove -command \
GetQuery
menubutton .mbar.level -bitmap @./bitmaps/level.xbm -relief groove -menu \
.mbar.level.menu
pack .mbar.file .mbar.algo .mbar.search .mbar.query .mbar.level -side left
```

```
menu .mbar.file.menu
.mbar.file.menu add command -label "HTML Document" -command {  \
 set filetype html
 DocumentSelect
}
.mbar.file.menu add command -label "SGML Document" -command {  \
 set filetype sgml
 DocumentSelect
}
menu .mbar.algo.menu
.mbar.algo.menu add cascade -label Symdifference -menu .mbar.algo.menu.sub1
.mbar.algo.menu add separator
.mbar.algo.menu add command -label Difference -command { \
     Comp1 difference
}
.mbar.algo.menu add command -label Union -command { \
     Comp1 union
}
.mbar.algo.menu add command -label Intersection -command { \
     Comp1 intersection
}
.mbar.algo.menu add command -label Merge -command { \
     comp2 merge
}
.mbar.algo.menu add command -label Mergeable -command { \
     comp2 mergeable
}
menu .mbar.search.menu
.mbar.search.menu add command -label Newest-Document -command { \
     comp2 newest
}
.mbar.search.menu add command -label Oldest-Document -command { \
   comp2 oldest
}
.mbar.search.menu add command -label Nearest-Neighbor -command { \
  comp2 nearestneighbor
}
.mbar.search.menu add command -label Furthest-Neighbor -command { \
    comp2 furthestneighbor
}
menu .mbar.level.menu
.mbar.level.menu add command -label Normal -command { \
set level 1
}
.mbar.level.menu add command -label Section -command { \
set level 2
}
```

```
.mbar.level.menu add command -label Paragraph -command { \
set level 3
}
menu .mbar.algo.menu.sub1
.mbar.algo.menu.sub1 add command -label Symdifference -command { \
Comp1  symdifference
}
.mbar.algo.menu.sub1 add command -label "Symdifference & Diff" \
-command {
Comp1  symdifference
}
.mbar.algo.menu.sub1 add command -label "Symdifference & Sdiff" \
-command {
Comp1  symdifference
}
}
```

```
 ------------------------------------------------------------------------
|                                                                        |
| Function: CreatListbox                                                 |
|                                                                        |
| This function creats list box as per given arguments.                  |
|________________________________________________________________________|
```

```
proc CreatListbox { parent args } {
frame $parent
eval {  listbox $parent.list -yscrollcommand [ list $parent.sy set ] \
-xscrollcommand [ list $parent.sx set] } $args
scrollbar $parent.sy -orient vertical \
-command [ list $parent.list yview]
frame $parent.bottom
scrollbar $parent.sx -orient horizontal \
-command [ list $parent.list xview]
set pad [ expr [$parent.sy cget -width] + 2* \
( [$parent.sy cget -bd] + [$parent cget -highlightthickness]\
 ) ]
frame $parent.pad -width $pad -height $pad
return $parent
}
```

```
 ------------------------------------------------------------------------
|                                                                        |
| Function: DocumentSelect                                               |
|                                                                        |
| This function creats window for document selection                     |
| from any directry.                                                     |
|________________________________________________________________________|
```

```
proc DocumentSelect { } {
global selectedlist
global filetype
global pwdir
set selectedlist { }
toplevel .ds
wm title .ds "[string toupper $filetype] Document Selection"
set left [ CreatListbox .ds.left -background #feeacf -width 25 \
-setgrid true]
set right [ CreatListbox .ds.right -background #feeacf -width 25 \
-setgrid true]
frame .ds.bot
pack .ds.bot -side bottom -fill x
pack $left $right -side left -expand true -fill both
pack $left.bottom -side bottom -fill x
pack $left.pad -in $left.bottom -side right
pack $left.sx -in $left.bottom -side bottom -fill x
pack $left.sy -side right -fill y
pack $left.list  -side left -fill both -expand true
pack $left $right -side left -expand true -fill both
pack $right.bottom -side bottom -fill x
pack $right.pad -in $right.bottom -side right
pack $right.sx -in $right.bottom -side bottom -fill x
pack $right.sy -side right -fill y
pack $right.list  -side left -fill both -expand true
button .ds.bot.ok -text "   OK    " -command { Okay }
button .ds.bot.clear -text "Clear All"  -command  {
global OkFlag
.ds.right.list delete 0 end
set OkFlag 0
}
button .ds.bot.quit -text "   Quit  " -command  { DocSelQuit }
pack .ds.bot.ok .ds.bot.clear .ds.bot.quit -side left -padx 15m \
-expand true
cd $pwdir
if { $pwdir == "/" } {
set pwdir ""
}
foreach f [ exec /bin/ls -a [pwd] ] {
if { $filetype == "html" } {
  if { [file extension $f] == ".htm" || [file extension $f] \
== ".html" } {
$left.list insert end $pwdir/$f
    }
}
if { $filetype == "sgml" } {
$left.list insert end $pwdir/$f
```

```
}

}
bind $left.list <ButtonRelease-1> [ list GoToDir $left ]
bind $right.list <ButtonPress-1> \
{ ListSelectStart .ds.right.list %y }
bind $right.list <B1-Motion> \
{ ListSelectExtend .ds.right.list %y }
bind $right.list <ButtonRelease-1> \
{ ListDeleteEnd .ds.right.list %y }
bind $right.list <ButtonPress-3> \
{ ListSelectStart .ds.right.list %y }
bind $right.list <B3-Motion> \
{ ListSelectExtend .ds.right.list %y }
bind $right.list <ButtonRelease-3> \
{ ViewDoc .ds.right.list %y }
bind $left.list <ButtonPress-3> \
{ ListSelectStart .ds.left.list %y }
bind $left.list <B3-Motion> \
{ ListSelectExtend .ds.left.list %y }
bind $left.list <ButtonRelease-3> \
{ ViewDoc .ds.left.list %y }

}
```

```
-------------------------------------------------------------------------
|                                                                       |
| Function: ListSelectStart, ListSelectExtend, ListDeleteEnd            |
|                                                                       |
| These functions are used to keep track of mouse movements             |
|-----------------------------------------------------------------------|
```

```
proc ListSelectStart { w y } {
$w select anchor [ $w nearest $y ]
}
proc ListSelectExtend { w y } {
$w select set anchor [ $w nearest $y ]
}
proc ListDeleteEnd { w y } {
global OkFlag
$w select set anchor [ $w nearest $y ]
foreach i [lsort -decreasing [ $w curselection ] ] {
 $w delete $i
}
$w selection clear 0 end
set OkFlag 0
}
```

```
#-------------------------------------------------------------------------
#                                                                         |
# Function: GoToDir, Okay, DocSelQuit                                     |
#                                                                         |
# These functions are used to navigate to different directories           |
#-------------------------------------------------------------------------#

proc GoToDir { left  } {
global OkFlag
set ck [ $left.list get  [ $left.list curselection ] ]
$left.list selection clear 0 end
if { [ file isdirectory $ck ]} {
cd $ck
if { [ file readable $ck ] } {
$left.list delete 0 end
} else {
     tk_dialog .err { Output File Reading Error } \
     "Read permission denied." warning 0 OK
return
}
set pwdir [pwd]
if { $pwdir == "/" } {
set pwdir ""
}
foreach f [ exec /bin/ls -a [pwd] ] {
$left.list insert end $pwdir/$f
}
} else {
set check [ lsearch [.ds.right.list get 0 end] $ck ]
if { $check == -1 } {
.ds.right.list insert end $ck
set OkFlag 0
} else {
     tk_dialog .err { Output File Reading Error } \
    "This document is already selected." warning 0 OK
}
}
}

proc Okay { } {
global selectedlist
global OkFlag
set selectedlist [ .ds.right.list get 0 end ]
set OkFlag 1
}
```

```
proc DocSelQuit { } {
global selectedlist
global OkFlag
set selectedlist [ .ds.right.list get 0 end ]
destroy .ds
}
```

```
 ------------------------------------------------------------------------
|                                                                        |
| Function: GetQuery                                                     |
|                                                                        |
| These functions creats window to write queries                         |
|________________________________________________________________________|
```

```
proc GetQuery { } {
toplevel .q
wm title .q "Query"
frame .q.select
pack .q.select -side top -fill x
button .q.select.select -text SELECT -relief raised -width 7  \
-command [ list .q.select.data delete 1.0 end]
pack .q.select.select -side left
text .q.select.data -relief sunken -height 1 -width 30 -wrap none
pack .q.select.data -side left -expand true -fill x
frame .q.from
pack .q.from -side top -fill x
button .q.from.from -text "   FROM" -relief raised -width 7 \
-command [ list .q.from.data delete 1.0 end]
pack .q.from.from -side left
text .q.from.data -relief sunken -height 1 -width 30 -wrap none
pack .q.from.data -side left -expand true -fill x
frame .q.where
pack .q.where -side top -expand true -fill both
button .q.where.where -text " WHERE" -relief raised -width 7  \
-command [ list .q.where.data delete 1.0 end]
pack .q.where.where -side left
text .q.where.data -relief sunken -height 3 -width 30 -wrap none
pack .q.where.data -side left -expand true -fill both
frame .q.tmpl1
pack  .q.tmpl1 -side top -fill x
button .q.tmpl1.b1 -text Q1 -relief raised -width 4 -command {
  .q.select.data delete 1.0 end
  .q.select.data insert 1.0 "*"
  .q.from.data delete 1.0 end
  .q.from.data insert 1.0 "docset1.set"
  .q.where.data delete 1.0 end
  .q.where.data insert 1.0 ""
```

```
}
button .q.tmpl1.b2 -text Q2 -relief raised -width 4 -command {
  .q.select.data delete 1.0 end
  .q.select.data insert 1.0 "xx"
  .q.from.data delete 1.0 end
  .q.from.data insert 1.0 "docset1.set"
  .q.where.data delete 1.0 end
  .q.where.data insert 1.0 "a IN docset1.set\nAND xx =  \
SYMDIFFERENCE(a, doc1.sgm)"
}

button .q.tmpl1.b3 -text Q3 -relief raised -width 4 -command {
  .q.select.data delete 1.0 end
  .q.select.data insert 1.0 "a"
  .q.from.data delete 1.0 end
  .q.from.data insert 1.0 "docset1.set"
  .q.where.data delete 1.0 end
  .q.where.data insert 1.0 "a IN docset1.set\nAND xx =  \
SYMDIFFERENCE(a, doc1.sgm)"
}

button .q.tmpl1.b4 -text Q4 -relief raised -width 4 -command {
  .q.select.data delete 1.0 end
  .q.select.data insert 1.0 "a"
  .q.from.data delete 1.0 end
  .q.from.data insert 1.0 "docset1.set"
  .q.where.data delete 1.0 end
  .q.where.data insert 1.0 "a IN docset1.set\nAND xx =  \
SYMDIFFERENCE(a, doc1.sgm)\nAND (SIZE(xx) > 1 and SIZE(xx) < 4)"
}

button .q.tmpl1.b5 -text Q5 -relief raised -width 4 -command {
  .q.select.data delete 1.0 end
  .q.select.data insert 1.0 "t"
  .q.from.data delete 1.0 end
  .q.from.data insert 1.0 "docset1.set, docset2.set"
  .q.where.data delete 1.0 end
  .q.where.data insert 1.0 "a IN docset1.set\nAND b IN  \
docset2.set\nAND t = DIFFERENCE (a, b)"
}
for {set i 1} {$i < 6} {incr i 1} {
  pack .q.tmpl1.b$i -side left -expand true
}
frame .q.tmpl2
pack  .q.tmpl2 -side top -fill x
button .q.tmpl2.b6 -text Q6 -relief raised -width 4 -command {
  .q.select.data delete 1.0 end
```

```
   .q.select.data insert 1.0 "t"
   .q.from.data delete 1.0 end
   .q.from.data insert 1.0 "docset1.set"
   .q.where.data delete 1.0 end
   .q.where.data insert 1.0 "t = intersection (docset1.set)"
}
button .q.tmpl2.b7 -text Q7 -relief raised -width 4 -command {
   .q.select.data delete 1.0 end
   .q.select.data insert 1.0 "m"
   .q.from.data delete 1.0 end
   .q.from.data insert 1.0 "memoset.set"
   .q.where.data delete 1.0 end
   .q.where.data insert 1.0 "m IN memoset.set\nAND DIST(m, memo1.sgm) \
 = MIN (DIST(u, memo1.sgm) WHERE u IN memoset.set)"
}
button .q.tmpl2.b8 -text Q8 -relief raised  -width 4 -command {
   .q.select.data delete 1.0 end
   .q.select.data insert 1.0 "m"
   .q.from.data delete 1.0 end
   .q.from.data insert 1.0 "memoset.set"
   .q.where.data delete 1.0 end
   .q.where.data insert 1.0 "a IN memoset.set\nAND m =  \
 nearest_neighbor (a, memoset.set)"
}
button .q.tmpl2.b9 -text Q9 -relief raised  -width 4 -command {
   .q.select.data delete 1.0 end
   .q.select.data insert 1.0 "k"
   .q.from.data delete 1.0 end
   .q.from.data insert 1.0 "docset1.set"
   .q.where.data delete 1.0 end
   .q.where.data insert 1.0 "k = furthest_neighbor  \
(doc1.sgm, docset1.set)"
}
button .q.tmpl2.b10 -text Q10 -relief raised  -width 4 -command {
   .q.select.data delete 1.0 end
   .q.select.data insert 1.0 "OLDEST"
   .q.from.data delete 1.0 end
   .q.from.data insert 1.0 "docset1.set"
   .q.where.data delete 1.0 end
   .q.where.data insert 1.0 ""
}
for {set i 6} {$i < 11} {incr i 1} {
  pack .q.tmpl2.b$i -side left -expand true
}
frame .q.run
pack .q.run -side bottom -fill x
button .q.run.run -text Run -relief raised  -width 5  \
```

```
-command {RunQuery}
button .q.run.quit -text Quit -relief raised  -width 5 -command {
destroy .q
}
pack .q.run.run .q.run.quit -side right  -padx 10 -pady 10  \
-expand true
}
```

```
 ------------------------------------------------------------------------
|                                                                        |
| Function: RunQuery                                                     |
|                                                                        |
| These functions passes queries to backend program                      |
| and displays the result.                                               |
|________________________________________________________________________|
```

```
proc RunQuery { } {
#save SQL in file
if { [ winfo exists .d.list ] } {
.d.list configure -state normal
.d.list delete 1.0 end
} else {
set fileId [open GetQuery w+ ]
puts $fileId "SELECT "
puts $fileId [ .q.select.data get 1.0 end ]
set where [ .q.from.data get 1.0 end ]
if { [ llength $where ] > 0 } {
puts $fileId "FROM "
puts $fileId [ .q.from.data get 1.0 end ]
}
set where [ .q.where.data get 1.0 end ]
if { [ llength $where ] > 0 } {
puts $fileId "WHERE "
puts $fileId $where
}
puts $fileId ";"
close $fileId
catch [ exec ./dql_parse < GetQuery >& QueryOPFile ]
if { ![ file exists QueryOPFile ] } {
return
}
toplevel .d
wm title .d "Query Results"
frame .d.quit
pack .d.quit -side top -fill x
button .d.quit.quite -text Quit -relief raised  -command {
destroy .d
catch [ exec rm GetQuery ]
```

```
catch [ exec rm QueryOPFile ]
}
pack .d.quit.quite -side right -fill x
text .d.list -relief sunken -bd 3  -yscrollcommand  ".d.scroll  \
 set "-xscrollcommand ".d.xscroll set" -height 30 -width 80 -wrap none
scrollbar .d.scroll -command ".d.list yview"
scrollbar .d.xscroll -command ".d.list xview" -orient horizontal
pack .d.scroll -side right -fill y
pack .d.xscroll -side bottom -fill x
pack .d.list -expand true -fill both
}
.d.list delete 1.0 end
set f [open QueryOPFile]
while { ![eof $f]} {
      set line "[gets $f ]\n"
      .d.list insert end $line
      }
close $f
.d.list configure -state disabled
return
}
```

```
 ------------------------------------------------------------------------
|                                                                        |
| Function: ViewDoc, DisplayDoc, DisplayDocComp2, texscroll              |
|                                                                        |
| These functions are used to display the document.                      |
|________________________________________________________________________|
```

```
proc ViewDoc { w y } {
$w select set anchor [ $w nearest $y ]
foreach i [lsort -decreasing [ $w curselection ] ] {
set file [ $w get $i ]
if { [ file isdirectory $file ] } {
continue
}
set l [ split $file . ]
set t [ join $l # ]
if { [ winfo exists .$t] } {
continue
}
DisplayDoc $t $file
}
$w selection clear 0 end
}

proc DisplayDoc { t file } {
```

```
set bkgcolor #ffe3d2
toplevel .$t
frame .$t.frm -background #ffd2b7
pack .$t.frm -side top -fill x
button .$t.frm.quit -text Quit -relief raised -background #ffbb93 \
-command [list destroy .$t]
pack .$t.frm.quit -side right  -padx 0 -pady 0
wm title .$t "Document: $file "
set txv  [text .$t.list1 -relief sunken -bd 2 -bg $bkgcolor \
 -yscrollcommand " .$t.scroll1 set" -xscrollcommand ".$t.xscroll1 set"]
scrollbar .$t.scroll1 -command ".$t.list1 yview"
scrollbar .$t.xscroll1 -command ".$t.list1 xview" -orient horizontal
pack .$t.scroll1 -side right -fill y
pack .$t.xscroll1 -side bottom -fill x
pack .$t.list1
.$t.list1 delete 1.0 end
set f [open $file ]
while { ![eof $f]} {
set line "[gets $f ]\n"
.$t.list1 insert end $line
}
.$t.list1 configure -state disabled
close $f
return
}

proc DisplayDocComp2 { t file } {
set bkgcolor #ffe3d2
toplevel .$t
frame .$t.frm -background #ffd2b7
pack .$t.frm -side top -fill x
button .$t.frm.quit -text Quit -relief raised -background #ffbb93 \
 -command [list destroy .$t]
pack .$t.frm.quit -side right  -padx 0 -pady 0
wm title .$t "$t"
set txv  [text .$t.list1 -relief sunken -bd 2 -bg $bkgcolor \
 -yscrollcommand " .$t.scroll1 set" -xscrollcommand ".$t.xscroll1 set"]
scrollbar .$t.scroll1 -command ".$t.list1 yview"
scrollbar .$t.xscroll1 -command ".$t.list1 xview" -orient horizontal
pack .$t.scroll1 -side right -fill y
pack .$t.xscroll1 -side bottom -fill x
pack .$t.list1
.$t.list1 delete 1.0 end
set f [open $file ]
while { ![eof $f]} {
set line "[gets $f ]\n"
.$t.list1 insert end $line
```

```
}
.$t.list1 configure -state disabled
close $f
return
}

proc texscroll { file  line } {
if { $line } {
set l [ split $file . ]
set t [ join $l # ]
if { ![ winfo exists .$t] } {
    DisplayDoc  $t $file
      }
.$.list1 yview [ expr $line - 1 ]
}
return
}
```

```
 ---------------------------------------------------------------------------
|                                                                           |
| Function: Comp1, Comp2, Multcomp                                          |
|                                                                           |
| These function are used to display results in hypertext form.             |
|___________________________________________________________________________|
```

```
proc Comp1 { operator } {
global OP
global OkFlag
global filetype
global level
global selectedlist
if { $OkFlag == 0 } {
tk_dialog .err { Document Selection Error } \
"In Document Selection Window, select your favoured  \
documents and press OK to confirm your selection." warning 0 OK
return
}
set TotalSelectDoc [ llength $selectedlist ]
if { [ string compare $operator "symdifference" ] == 0 &&  \
$TotalSelectDoc != 2  } {
        tk_dialog .err { Document Selection Error } \
 "Total no of document selected: $TotalSelectDoc !!     \
 For Symdifference two documents requires. \
 Select again." warning 0 OK
 return
}
set SelectedFiles ""
```

```
foreach doc $selectedlist {
set SelectedFiles "$SelectedFiles [ file tail $doc ]"
}
puts $SelectedFiles
set file $operator
foreach doc [ lsort $selectedlist ] {
set file $file.$doc
}
set t [ split $file . ]
set l [ join $t # ]
if { [ winfo exists .$l] } {
puts "Already This comparision exists"
return
}
incr OP 1
set para1 [lindex $selectedlist 0 ]
set para2 [lindex $selectedlist 1 ]
set para3 [lindex $selectedlist 2 ]

switch $operator {
    symdifference {
if { $filetype == "sgml"} {
catch [ eval "exec ./tdsgml -d $SelectedFiles  > op$OP" ]
} else {
catch [ eval "exec ./tdsgml -p ./html_dtd/catalog  \
./html_dtd/html.decl  $SelectedFiles  > op$OP" ]
}
}
   difference {
if { $level == 1 } {
catch [ eval "exec ./tdsgml -s $SelectedFiles  > op$OP" ]
} elseif { $level == 2 } {
catch [ eval "exec ./tdsgml -sec  $SelectedFiles  > op$OP" ]
} elseif { $level == 3 } {
catch [ eval "exec ./tdsgml -par  $SelectedFiles  > op$OP" ]
}
}
   union {
if { $level == 1 } {
catch [ eval "exec ./tdissgml -k 0 $SelectedFiles  > op$OP" ]
} elseif { $level == 2 } {
catch [ eval "exec ./tdissgml -sec -k 0 $SelectedFiles  >  \
op$OP" ]
} elseif { $level == 3} {
catch [ eval "exec ./tdissgml -par -k 0 $SelectedFiles  >  \
op$OP" ]
}
```

```
}
   intersection  {
catch [ eval "exec ./tdissgml -i -k 0 $SelectedFiles  >  \
op$OP" ]
}
   default {
puts "Wrong operator"
return
        }
}
toplevel .$l
wm title .$l "[ string toupper $operator]"
frame .$l.frm
pack .$l.frm -side top -fill x
pack .$l.frm.quit -side right -ipadx 2m
text .$l.list -relief sunken -bd 3  -yscrollcommand  ".$l.scroll \
set" -xscrollcommand ".$l.xscroll set" -width 75 -wrap none \
-font *-*-bold-*-*-10-* -background #feeacf
scrollbar .$l.scroll -command ".$l.list yview"
scrollbar .$l.xscroll -command ".$l.list xview" -orient horizontal
pack .$l.scroll -side right -fill y
pack .$l.xscroll -side bottom -fill x
pack .$l.list -expand 1 -fill both
bind .$l.list <Enter>  [list .$l.list config -cursor hand2]
.$l.list delete 1.0 end
set linecounter 0
set f [open op$OP ]
while { ![eof $f]} {
set line [gets $f ]
.$l.list insert end "$line\n"
incr linecounter 1
if { [regexp  "Line No"  $line ] } {
     break
   }
}
set line [gets $f]
set ClrInt1 6
set ClrInt2 6
set part 0

while { ![eof $f]} {
set line [gets $f ]
set line "$line\n"
.$l.list insert end $line
incr linecounter 1
if  { ![ string compare "----------" [string range $line 25 34]] } {
    MultComp  $f $linecounter $l $part
```

```
    break
    }
set array "$line"
set s [string range $line 33 34]
if { ![string compare " =" $s]} {
    incr part 1
    set doc1line [lindex $array 0]
    set i [lsearch $array  "="]
    incr i +1
    set doc2line [lindex $array $i]
    set ClrInt1 [lindex $array 1]
    set ClrInt2 [lindex $array [ incr i +1 ]]
    set clr1 [ ColorInt black $ClrInt1 ]
    set clr2 [ ColorInt black $ClrInt2 ]
} elseif { ![string compare " |" $s]} {
    incr part 1
    set doc1line [lindex $array 0]
    set i [lsearch $array  "|"]
    incr i +1
    set doc2line [lindex $array $i]
    set ClrInt1 [lindex $array 1]
    set ClrInt2 [lindex $array [ incr i +1 ]]
    set clr1 [ ColorInt red $ClrInt1 ]
    set clr2 [ ColorInt red $ClrInt2 ]
} elseif { ![string compare " <" $s]} {
    incr part 1
    set doc1line [lindex $array 0]
    set doc2line 0
    set ClrInt2 6
    set ClrInt1 [lindex $array 1 ]
    set clr1 [ ColorInt purple  $ClrInt1 ]
    set clr2 [ ColorInt black $ClrInt2 ]
} elseif { ![string compare " >" $s]} {
    incr part 1
    set i [lsearch $array  ">"]
    incr i +1
    set doc2line [lindex $array $i]
    set doc1line 0
    set ClrInt1 6
    set ClrInt2 [lindex $array [ incr i +1 ]]
    set clr1 [ ColorInt black $ClrInt1 ]
    set clr2 [ ColorInt blue $ClrInt2 ]
} elseif { ![string compare " X" $s]} {
    incr part +1
    set doc1line [lindex $array 0]
    set i [lsearch $array  "X"]
    incr i +1
```

```
    set doc2line [lindex $array $i]
    set ClrInt1 [lindex $array 1]
    set ClrInt2 [lindex $array [ incr i +1 ]]
    if { ![string compare "X" $doc1line ] } {
set doc1line 0
set ClrInt1 0
set ClrInt2 [lindex $array [ incr i +1 ]]
     }
    if { ![string compare " " $doc2line ] } {
set doc2line 0
set ClrInt2 0
set ClrInt1 [lindex $array 1 ]
     }
    set clr1 [ ColorInt green $ClrInt1 ]
    set clr2 [ ColorInt green $ClrInt2 ]
}
if { ![ regexp {^[0-9]+$} $ClrInt1] } {
   set ClrInt1 6
   } elseif { ![ regexp {^[0-9]+$} $ClrInt2] } {
   set ClrInt2 6
}
.$1.list tag add big1$part $linecounter.0 $linecounter.34
.$1.list tag add big2$part $linecounter.35 $linecounter.end
.$1.list tag configure big1$part -foreground $clr1 -relief flat
.$1.list tag bind big1$part <Button-1> [list texscroll [ lindex \
$selectedlist 0 ]  $doc1line ]
.$1.list tag configure big2$part -foreground $clr2 -relief flat
.$1.list tag bind big2$part <Button-1> [list texscroll [ lindex \
$selectedlist 1 ]  $doc2line ]
} # while ends here
set level 1
.$1.list configure -state disabled
catch [ exec  rm op$OP ]
}

proc comp2 { operator } {
global OP
global OkFlag
global selectedlist
if { $OkFlag == 0 } {
tk_dialog .err { Document Selection Error } \
"In Document Selection Window, select documents and press \
OK to confirm your  selection." warning 0 OK
return
}
set SelectedFiles ""
foreach doc $selectedlist {
```

```
set SelectedFiles "$SelectedFiles [ file tail $doc ]"
}
puts $SelectedFiles
set file $operator
foreach doc [ lsort $selectedlist ] {
set file $file.$doc
}
set t [ split $file . ]
set l [ join $t # ]
if { [ winfo exists .$l] } {
puts "Already This comparision exists"
return
}
incr OP 1
set TotalSelectDoc [ llength $selectedlist ]
switch $operator {
   merge  {
if { $TotalSelectDoc == 2 && ![ IsSetFileThere $selectedlist] \
   } {
catch [ eval "exec ./tdsgml -n $SelectedFiles  > \
 op$OP" ]
set l [ split op$OP . ]
set t [ join $l # ]
if { [ winfo exists .$t] } {
puts "This is existing nearest-neighbour"
return
}
DisplayDocComp2 $operator op$OP
} else  {
    tk_dialog .err { Document Selection Error } \
    "Total no of document selected: $TotalSelectDoc !!    \
    For Merge two regular file requires. \
    Select again." warning 0 OK
return
}
}
  mergeable {
if { $TotalSelectDoc == 2 && ![ IsSetFileThere $selectedlist] \
   } {
catch [ eval "exec ./tdsgml -m $SelectedFiles  >  \
op$OP" ]
set l [ split op$OP . ]
set t [ join $l # ]
if { [ winfo exists .$t] } {
puts "This is existing nearest-neighbour"
return
}
```

```
DisplayDocComp2 $operator op$OP
} else  {
    tk_dialog .err { Document Selection Error } \
    "Total no of document selected: $TotalSelectDoc !!    \
    For Mergeable two regular file requires. \
    Select again." warning 0 OK
return
}
}
  newest {
if { $TotalSelectDoc == 1 && [IsSetFileThere $selectedlist] \
   } {
catch [ eval "exec ./newest  $SelectedFiles  >  \
op$OP" ]
set l [ split op$OP . ]
set t [ join $l # ]
if { [ winfo exists .$t] } {
puts "This is existing newest"
return
}
DisplayDocComp2 $operator op$OP
} else  {
    tk_dialog .err { Document Selection Error } \
    "Total no of document selected: $TotalSelectDoc !!    \
    For Newest operator one set file requires. \
    Select again." warning 0 OK
return
}
}
  oldest {
if { $TotalSelectDoc == 1 && [ IsSetFileThere $selectedlist] \
   } {
catch [ eval "exec ./oldest $SelectedFiles  > op$OP" ]
set l [ split op$OP . ]
set t [ join $l # ]
if { [ winfo exists .$t] } {
puts "This is existing oldest"
return
}
DisplayDocComp2 $operator op$OP
} else  {
    tk_dialog .err { Document Selection Error } \
    "Total no of document selected: $TotalSelectDoc !!    \
    For Oldest operator one set file requires. \
    Select again." warning 0 OK
return
}
```

```
}
  nearestneighbor {
if { $TotalSelectDoc == 2 && [ IsSetFileThere $selectedlist] \
 && [IsRegularFileThere  $selectedlist ] } {
catch [ eval "exec ./nearest  $SelectedFiles  > op$OP" ]
set l [ split op$OP . ]
set t [ join $l # ]
if { [ winfo exists .$t] } {
puts "This is existing nearest-neighbour"
return
}
DisplayDocComp2 $operator op$OP
} else  {
    tk_dialog .err { Document Selection Error } \
    "Total no of document selected: $TotalSelectDoc !!    \
    For Nearest-Neighbour one regular file and one set file \
 requires. Select again." warning 0 OK
return
}
}
  furthestneighbor {
if { $TotalSelectDoc == 2 && [ IsSetFileThere $selectedlist] \
&& [IsRegularFileThere  $selectedlist ] } {
catch [ eval "exec ./furthest  $SelectedFiles  > \
op$OP" ]
set l [ split op$OP . ]
set t [ join $l # ]
if { [ winfo exists .$t] } {
puts "This is existing nearest-neighbour"
return
}
DisplayDocComp2 $operator op$OP
} else  {
    tk_dialog .err { Document Selection Error } \
    "Total no of document selected: $TotalSelectDoc !!    \
    For Nearest-Neighbour one regular file and one set file \
 requires. Select again." warning 0 OK
return
}
}
  substructure {
if { $TotalSelectDoc == 2 && ![ IsSetFileThere $selectedlist] \
   } {
catch [ eval "exec ./tdsgml -sub $SelectedFiles  > \
op$OP" ]
set l [ split op$OP . ]
set t [ join $l # ]
```

```
 if { [ winfo exists .$t] } {
 puts "This is existing substructure"
 return
 }
 DisplayDocComp2 $operator op$OP
 } else  {
     tk_dialog .err { Document Selection Error } \
     "Total no of document selected: $TotalSelectDoc !!     \
     For SubStructure two regular file requires. \
     Select again." warning 0 OK
 return
 }
 }
   superstructure {
 if { $TotalSelectDoc == 2 && ![ IsSetFileThere $selectedlist] \
    } {
 catch [ eval "exec ./tdsgml -super $SelectedFiles  > \
 op$OP" ]
 set l [ split op$OP . ]
 set t [ join $l # ]
 if { [ winfo exists .$t] } {
 puts "This is existing superstructure"
 return
 }
 DisplayDocComp2 $operator op$OP
 } else  {
     tk_dialog .err { Document Selection Error } \
     "Total no of document selected: $TotalSelectDoc !!     \
     For SuperStructure two regular file requires. \
     Select again." warning 0 OK
return
}
}
    default         { puts "Wrong operator"
     return
}
}
}

proc MultComp { f linecounter l part } {
global selectedlist
set clr1 red
set clr2 red
set ClrInt1 6
set ClrInt2 6
set indx 1
set line "-------------------------------------------------"
```

```
while { ![eof $f]} {
if  { ![ string compare "----------" [string range $line 25 34]] } {
    while { ![eof $f]} {
       set line [gets $f ]
       if { [regexp  "Line No"  $line ] } {
    incr indx +1
    set line [gets $f ]
    break
       } else {
    set line "$line\n"
   .$l.list insert end $line
    incr linecounter 1
       }
    }
}
if { [eof $f] } { break
   }
set line [gets $f ]
incr linecounter 1
set line "$line\n"
.$l.list insert end $line
set array "$line"
set s [string range $line 33 34]
if { ![string compare " =" $s]} {
    incr part 1
    set doc1line [lindex $array 0]
    set i [lsearch $array  "="]
    incr i +1
    set doc2line [lindex $array $i]
    set ClrInt1 [lindex $array 1]
    set ClrInt2 [lindex $array [ incr i +1 ]]
    set clr1 [ ColorInt black $ClrInt1 ]
    set clr2 [ ColorInt black $ClrInt2 ]
} elseif { ![string compare " |" $s]} {
    incr part 1
    set doc1line [lindex $array 0]
    set i [lsearch $array  "|"]
    incr i +1
    set doc2line [lindex $array $i]
    set ClrInt1 [lindex $array 1]
    set ClrInt2 [lindex $array [ incr i +1 ]]
    set clr1 [ ColorInt red $ClrInt1 ]
    set clr2 [ ColorInt red $ClrInt2 ]
} elseif { ![string compare " <" $s]} {
    incr part 1
    set doc1line [lindex $array 0]
    set doc2line 0
```

```
    set ClrInt2 6
    set ClrInt1 [lindex $array 1 ]
    set clr1 [ ColorInt purple  $ClrInt1 ]
    set clr2 [ ColorInt black $ClrInt2 ]
} elseif { ![string compare " >" $s]} {
    incr part 1
    set i [lsearch $array  ">"]
    incr i +1
    set doc2line [lindex $array $i]
    set doc1line 0
    set ClrInt1 6
    set ClrInt2 [lindex $array [ incr i +1 ]]
    set clr1 [ ColorInt black $ClrInt1 ]
    set clr2 [ ColorInt blue $ClrInt2 ]
} elseif { ![string compare " X" $s ] } {
    incr part 1
    set doc1line [lindex $array 0]
    set i [lsearch $array  "X"]
    incr i +1
    set doc2line [lindex $array $i]
    set ClrInt1 [lindex $array 1]
    set ClrInt2 [lindex $array [ incr i +1 ]]
    if { ![string compare "X" $doc1line ] } {
set doc1line 0
set ClrInt1 0
set ClrInt2 [lindex $array [ incr i +1 ]]
    }
    if { ![string compare " " $doc2line ] } {
set doc2line 0
set ClrInt2 0
set ClrInt1 [lindex $array 1 ]
    }
    set clr1 [ ColorInt green $ClrInt1 ]
    set clr2 [ ColorInt green $ClrInt2 ]
}
if { ![ regexp {^[0-9]+$} $ClrInt1] } {
set ClrInt1 6
} elseif { ![ regexp {^[0-9]+$} $ClrInt2] } {
set ClrInt2 6
}
.$l.list tag add big1$part $linecounter.0 $linecounter.34
.$l.list tag add big2$part $linecounter.35 $linecounter.end
.$l.list tag configure big1$part -foreground $clr1 -relief flat
.$l.list tag configure big2$part -foreground $clr2 -relief flat
.$l.list tag bind big2$part <Button-1> [list texscroll [ lindex \
$selectedlist $indx ]  $doc2line ]
} # while ends here
```

```
return
}
```

```
 ------------------------------------------------------------------------
|                                                                        |
| Function: ColorInt                                                     |
|                                                                        |
| This function is used to display text in different                     |
| intensity of color.                                                    |
|________________________________________________________________________|
```

```
proc ColorInt { color int } {
switch $color {
   red {
  if { $int == 1 }      { return #ec3535
} elseif { $int == 2 } { return #ec1515
} elseif { $int == 3 } { return #ec0000
} elseif { $int == 4 } { return #d70000
} elseif { $int == 0 } { return #ff5151
} else { return #c60000
}
}
  green {
  if { $int == 0 }      { return #009b00
} elseif { $int == 1 } { return #008d00
} elseif { $int == 2 } { return #008200
} elseif { $int == 3 } { return #006f00
} elseif { $int == 4 } { return #006500
} else { return #005900
}
       }
  blue {
  if { $int == 1 }      { return #3e3eff
} elseif { $int == 2 } { return #1313ff
} elseif { $int == 3 } { return #0000e8
} elseif { $int == 4 } { return #0000c4
} elseif { $int == 0 } { return #6060ff
} else { return #000092
}
       }
   purple {
  if { $int == 1 }      { return #cc22cc
} elseif { $int == 2 } { return #b71eb7
} elseif { $int == 3 } { return #9e1b9e
} elseif { $int == 4 } { return #8a168a
} elseif { $int == 0 } { return #dc27dc
} else { return #701270
}
```

```
        }
    default {
   if { $int == 0 }      { return #444444
} elseif { $int == 1 } { return #444444
} elseif { $int == 2 } { return #222222
} elseif { $int == 3 } { return #222222
} elseif { $int == 4 } { return #111111
} else { return #111111
}
        }
}
}
```

```
-------------------------------------------------------------------------
|                                                                       |
| These functions are used to find type of the document.                |
|-----------------------------------------------------------------------|
```

```
proc IsSetFileThere { fileslist } {
foreach doc $fileslist {
if { [ file ext $doc ] == ".set"} { return 1
   }
}
return 0
}
proc IsRegularFileThere { fileslist } {
foreach doc $fileslist {
if { [ file ext $doc ] != ".set"} { return 1
    }
}
return 0
}
proc IsAllHtmlFile { fileslist } {
foreach doc $fileslist {
if { [ file ext $doc ] = ".html" || [ file ext $doc ] = \
".htm"} { continue
    }else {
tk_dialog .err { Document Selection Error } \
"All files should be html files. Select again." \
 warning 0 OK
return 0
    }
}
return 1
}
proc IsAllSgmlFile { fileslist } {
foreach doc $fileslist {
if { [ file ext $doc ] = ".sgml" || [ file ext $doc ] = \
```

```
".sgm"} { continue
   }else {
tk_dialog .err { Document Selection Error } \
"All files should be sgml files. Select again." \
 warning 0 OK
return 0
     }
}
return 1
}
```

REFERENCES

1. T. Cahill, M. G. Hinchey, and L. Relihan. Documents are programs. In *Proc. ACM SIGDOC*, Waterloo, Canada, October 1993.

2. C.-Y. Chang and J. T. L. Wang. Scientific data mining: A case study. In *Proceedings of the 8th International Conference on Software Engineering and Knowledge Engineering*, pages 100–107, Lake Tahoe, Nevada, June 1996.

3. S. S. Chawathe, A. Rajaraman, H. Garcia-Molina, and J. Widom. Change detection in hierarchically structured information. In *Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data*, pages 493–504, Montreal, Quebec, Canada, June 1996.

4. V. Christophides, S. Abiteboul, S. Cluet, and M. Scholl. From structured documents to novel query facilities. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 313–324, Minneapolis, Minnesota, May 1994.

5. C. Faloutsos and S. Christodoulakis. Signature files: An access method for documents and its analytical performance evaluation. *ACM Transactions on Office Information Systems*, 2(4):267–288, October 1984.

6. E. A. Fox, L. S. Heath, Q. F. Chen, and A. M. Daoud. Practical minimal perfect hash functions for large databases. *Communications of the ACM*, 35(1):105–121, January 1992.

7. C. F. Goldfarb. *The SGML Handbook*. Oxford University Press, New York, 1990.

8. A. Haake. CoVer: A contextual version server for hypertext applications. In *Proceedings of the 4th ACM Conference on Hypertext*, 1992.

9. E. V. Herwijnen. *Practical SGML*. Kluwer Academic Publishers, Boston, Massachusetts, 2 edition, 1994.

10. Mil-M-28001A. Markup requirements and generic style specification for electronic printed output and exchange of text. Department of Defense CALS Office, September 1990.

11. H. Moller. Versioning structured technical documentation. In *Proceedings of the Workshop on Versioning in Hypertext Systems*, 1994.

12. K. Osterbye. Structural and cognitive problems in providing version control for hypertext. In *Proceedings of the ACM Conference on Hypertext*, 1992.

13. C. V. Ramamoorthy and W. T. Tsai. Advances in software engineering. *IEEE Computer*, 29(10):47–58, October 1996.

14. J. T. L. Wang and C.-Y. Chang. Fast retrieval of electronic messages that contain mistyped words or spelling errors. *IEEE Transactions on Systems, Man, and Cybernetics* (forthcoming).

15. J. T. L. Wang, G. J. S. Chang, G.-W. Chirn, C.-Y. Chang, W. Wu, and F. Aljallad. A visualization tool for pattern matching and discovery in scientific databases. In *Proceedings of the 8th International Conference on Software Engineering and Knowledge Engineering*, pages 563–570, Lake Tahoe, Nevada, June 1996.

16. J. T. L. Wang, G.-W. Chirn, T. G. Marr, B. A. Shapiro, D. Shasha, and K. Zhang. Combinatorial pattern discovery for scientific data: Some preliminary results. In *Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data*, pages 115–125, Minneapolis, Minnesota, May 1994.

17. J. T. L. Wang, T. G. Marr, D. Shasha, B. A. Shapiro, and G.-W. Chirn. Discovering active motifs in sets of related protein sequences and using them for classification. *Nucleic Acids Research*, 22(14):2769–2775, 1994.

18. J. T. L. Wang, T. G. Marr, D. Shasha, B. A. Shapiro, G.-W. Chirn, and T. Y. Lee. Complementary classification approaches for protein sequences. *Protein Engineering*, 9(5):381–386, 1996.

19. J. T. L. Wang and P. A. Ng. TEXPROS: An intelligent document processing system. *International Journal of Software Engineering and Knowledge Engineering*, 2(2):171–196, June 1992.

20. J. T. L. Wang, K. Zhang, K. Jeong, and D. Shasha. A system for approximate tree matching. *IEEE Transactions on Knowledge and Data Engineering*, 6(4):559–571, August 1994.

21. S.-J. Yoo, P. B. Berra, Y. K. Lee, and K. Yoon. Version management in structured document retrieval systems. In *Proceedings of the 8th International Conference on Software Engineering and Knowledge Engineering*, pages 537–544, Lake Tahoe, Nevada, June 1996.

22. K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM Journal on Computing*, 18(6):1245–1262, December 1989.

23. K. Zhang, D. Shasha, and J. T. L. Wang. Approximate tree matching in the presence of variable length don't cares. *Journal of Algorithms*, 16(1):33–66, January 1994.