# XLSI – A Graphical User Interface for a Conceptual Retrieval System

Susan Clower Allen

Computer Science Department

CS-94-264 December 1994

# XLSI - A Graphical User Interface for a Conceptual Retrieval System

A Thesis

Presented for the

Master of Science Degree

The University of Tennessee, Knoxville

Susan Clower Allen

December 1994

**Acknowledgements**

I would like to thank my family for their patience throughout my thesis project. My husband, Jeff for allowing me to commit the required time and energy, and my daughter Hannah for her inspiration.

I would also like to thank Dr. Michael Berry, for his support and motivation. I appreciate his enthusiasm, patience and vast knowledge, which he so willingly shared. I would like to thank my thesis committee, Dr. Brad Vander Zanden and Dr. David Straight. I am grateful for the listening ears and helpful eyes of Rhonda MacIntyre. I would also like to thank Dr. Susan Dumais at Bellcore and Michael Littman at Brown University for their original LSI user interface and suggestions on features for XLSI.

## Abstract

In this thesis, XLSI, a Motif-based user interface, for utilizing Latent Semantic Indexing (LSI) retrieval techniques is presented. XLSI is a third generation Windows-based user interface that allows access to many kinds of textual material, for users of different levels of experience. The original LSI user interface, InfoSearch provided the incorporation of conceptual or fuzzy-based retrieval methodologies. The second generation user interface, XTDRS, provided basic windowing capabilities, utilized retrieval techniques based on LSI and was designed as an interface for novice X-windows users. As with its predecessors, XLSI utilizes the underlying LSI retrieval technologies and displays basic information required for query matching. However, as XTDRS and InfoSearch were designed for novice windows users, XLSI was designed to facilitate advanced user interaction and advanced informational display. XLSI extends the original information displayed by providing the user with the ability to retrieve previously-performed searches. This advanced information and any other information displayed by XLSI can be electronically exchanged with other users. XLSI can be compiled and executed on any workstation that supports the X-Window System Release 5 (X11R5) and the Open Software Foundation's (OSF) Motif 1.2.

# Contents

# List of Tables

# List of Figures

# Chapter 1

# Introduction

In this thesis, XLSI, a graphical user interface for latent semantic indexing (LSI) information retrieval is presented. XLSI can be used to retrieve information from a wide variety of textual materials, as well as from objects and services (e.g. electronic mail). The motivation of XLSI and pertinent background information is presented in this chapter. In Chapter 2, the design principles of XLSI are described. Chapter 3 describes how XLSI has been implemented. Chapter 4 provides a sample session utilizing XLSI. Chapter 5 describes the current computing platforms and future work for XLSI. Appendix A describes the interactive environment required to execute XLSI. Appendix B describes the customizable features of XLSI. Documentation on the XLSI software is given in Appendix C.

## 1.1 Motivation

XLSI provides a graphical user interface for the utilization of latent semantic indexing (LSI) in information retrieval. One of the initial design goals was to develop an interface based upon an industry standard windowing system. The X-Windows System Release 5 (X11R5) was chosen as the entry point into the X-Windows environment. To facilitate the development of XLSI, the Motif Version 1.2 libraries were selected as the X-based Toolkit. With X11R5 and Motif 1.2, XLSI could be *portable* across many computing platforms (UNIX and non-UNIX systems alike). Finally, for rapid prototyping, Visual Edge Software's UIM/X interface development tool was utilized.

In addition to portability, the X-Windows system provides a client/server architecture. The XLSI *engine* can execute on a high-performance workstation, while the interface is displayed on either a low cost workstation, an X-Terminal, or a personal computer running X-Windows emulation software.

XLSI was also designed to provide an intuitive interface to a wide range of users, including non-technical users, non-windows users and experienced windows users.

To provide this extensive degree of customization, XLSI incorporates user-defined resources. By allowing the end-user to define display characteristics, such as background colors or scrollbar locations, XLSI can be modified to support the particular skill level of the user. By default, the resources have been configured for the novice user.

Finally, XLSI should allow users to share both the information retrieved as well as the information that would indicate the success or failure of the underlying retrieval technique (LSI). The sharing of the information is implemented by adding an interface to the computer's electronic mail system. To provide additional information to the LSI-experienced users, XLSI implements the recording and displaying of previous search information.

## 1.2 Latent Semantic Indexing

In analyzing information retrieval methodologies, there exists a fundamental problem of trying to match words of queries with words of documents. Users want to retrieve information based on the meaning of the document, not necessarily the literal *keywords* within a document. An additional problem is that words may have more than one meaning, so words used in a user's query will match words in irrelevant documents. For example, the word chip may refer to a computer processing device (PA-RISC 7100) or a thin slice of a potato (a potato chip). Hence, a search with the word *chip* may retrieve documents related to food or computers. If the user is interested in computer technologies, a search with the word chip might not retrieve any relevant information. Latent Semantic Indexing (LSI) tries to overcome the problem of literal searches by treating the observed word to document association as an unreliable estimate of the larger number of words that could be associated with the document. Using mathematical models based upon the singular value decomposition (SVD), LSI can estimate this latent structure and remove the obscuring noise [DDF+90]. The closeness of documents is determined by the overall pattern of word usage so that documents can be grouped together regardless of the precise words used to describe them. In our above example, if the user was interested in computer technologies, most likely the documents concerning computers would have more *like* words than documents concerning food. Words, or terms, such as *electrical*, *instructions*, *performance*, *MIPS*, and *Fortran* could be common in computer-related documents, but not likely in documents on snack foods.

## 1.3 Relevance Feedback

With relevance feedback [SB90], words can exist in related documents, but not necessarily in a user's query. Continuing with the above example, if a user finds a document that contains information about Intel's Pentium processor, but actually wants information concerning performance characteristics of computers, they can use the entire Intel Pentium document as a query. Because relevance feedback-based queries better reflect the *context* of desired documents, a user may find information that did not contain the original keywords of the search (chip). Relevance feedback within LSI allows these documents to be clustered based upon the reuse of keywords within the documents, not necessarily within a user's search.

## 1.4 Ranking Relevant Documents

The original LSI process begins with a matrix of terms by documents. This matrix is derived from the original document(s) text so that each element is the frequency in which a particular term occurs in a given document. The latent semantic structure modeled by LSI requires a truncated singular value decomposition (SVD) of the term by document matrix [Ber92]. The truncated SVD can be viewed as a technique for deriving a set of uncorrelated factors. Each word (term) and document is represented by its vector of factor values (elements of scaled left and right singular values). When the user issues a query, the query is processed as a *pseudo-document*, i.e., the query is represented as a vector in the same multi-dimensional space containing the existing documents. The query vector is then compared to the conceptually-nearest documents. All documents comprising the database can be ordered with respect to *closeness* with the query. One measure of closeness for rank-ordering documents is the cosine between the vectors [Dum91]. The document corresponding to the highest cosine is simply the closest document in the $k$-dimensional subspace as determined by the truncated SVD of the original term-document matrix.

## 1.5 LSI Factors

As mentioned in Section 1.4, LSI requires a truncated singular value decomposition (SVD) of the term by document matrix. Each word (term) and document is represented by its vector of factor values (elements of left and right singular vectors) in $k$-space. The number of dimensions, $k$ is represented by the number of factors generated by the SVD. In an attempt to map *like* documents closer, users can reduce the number of factors or dimensions. By reducing the dimensions (factors), like documents *might* move closer together (as reflected by a higher cosine value). However, if the number of dimensions are too small, the *relevance* between documents may be forced[DDF$^+$90].

# Chapter 2

# Design Principles

XLSI is a third generation graphical user interface developed for the LSI application discussed in [DDF$^+$90]. The basis for the original LSI windows based interface was the MGR windowing system, which was developed internally and utilized at Bellcore. This interface was not based upon an industry standard window system or toolkit, therefore it was not portable. With XLSI, an initial design goal was to be portable across hardware platforms. By utilizing the X-Windows System Release 5 (X11R5) and the Motif 1.2 toolkits and libraries, XLSI can be ported to a wide variety of computing platforms.

One of the design principles of XLSI is to improve the users' ability to perform their information retrieval tasks. This requires understanding the users' skill level(s). An initial version of XLSI, XTDRS [BAM93], was designed to provide a basic interface to the LSI retrieval technologies. This version had a single user audience, librarians and library patrons at John C. Hodges Library on the campus of The University of Tennessee, Knoxville. Since this version was designed with a single audience, its usage was very simple, to provide very basic windows interfaces to the retrieval technologies. The features were limited by the experience level of the user community. In contrast, XLSI was designed for users with various experience levels.

In contrasting XLSI with the earlier interface designs for LSI, an easy method to input the search criteria did not exist. Additionally, the output (display) of the information was not in a format that the non-technical user could easily interpret. By providing a user-friendly interface, XLSI gives both the novice and the expert LSI user the capability to retrieve and display accurate information. Additionally, the early interfaces just displayed the information retrieved, without attempting to exploit conceptual or fuzzy search techniques. Through the enhanced features of XLSI, users can develop effective LSI search techniques.

Another design principle of XLSI is to enable users to accomplish searches that would otherwise be difficult or impossible. To accomplish this goal, XLSI's input interfaces provide the format for constructing a unique set of search criteria. In previous interfaces, the user could enter either keywords to be utilized in the search, or a title (which represents the entire document and thus a relevance feedback search). XLSI expands this search capability by allowing the user to enter keywords, titles (which represent the entire document) or a combination of both keywords and titles (documents) as search criteria. This feature allows users to retrieve information that they may have previously not been able to locate.

One challenge for LSI search techniques is the understanding of the relevance feedback searches. It is sometimes confusing to a user who is familiar with keyword-only searches. With XLSI, related documents which do not contain the exact keywords used in the query may be retrieved. XLSI attempts to educate the user of the underlying search process in its uniquely-labeled input interface as well as in the display (output) interface(s).

In previous LSI interfaces, the user's environment was a closed system with no capabilities of communicating with other LSI users. XLSI is designed for enhancing the users' ability to communicate with other users. Each individual interface in XLSI provides the user the capability to use electronic mail, print or save the associated textual output, thus allowing multiple users of the same documents to communicate the information they retrieved, the search criteria they used and the efficiency of their searches.

As mentioned earlier, XLSI is designed for users of different skill levels. Each user has unique preferences and needs. XLSI allows the user to have some control of the configuration of the interfaces. This level of customization is provided by utilizing the X-Window and Motif resources.

# Chapter 3

# Implementation

As stated previously, XLSI is implemented using X11R5 and Motif 1.2 libraries and toolkits. The following sections describe each individual interface implementation, as well as the specific implementation of the X11R5 and Motif 1.2 toolkits. Sample displays of the interfaces discussed in this chapter are found in Chapter 4.

## 3.1 Book Selection

The LSI application refers to a collection of documents as a book. Users need to choose which of these books they want to search, and since books can be modified, XLSI needed to provide a mechanism for adding, deleting and modifying documents within books without recompiling the application.

To display the currently configured books, XLSI utilizes the Motif `scrolledList` widget. The `scrolledList` widget allows the user to select the book that contains the documents that they would like to search. Selection occurs when the user double clicks on an item in the list. After the user has chosen a book, the XLSI application retrieves the book title from the scrolledList widget. The book title is then passed to the XLSI processing code to check for the presence of all the required LSI generated database files and XLSI database files.

To facilitate the addition of new books, the modification of the current book titles, and the deletion of old books, the XLSI application makes extensive use of resource files. For example, the resources associated with the `scrolledList` widget for book titles are listed below.

```
*.scrolledList1.itemCount:  5
*.scrolledList1.items: Collier Condensed Encyclopedia,
Gospels of the Bible, Knoxville News Paper, Netlib,
Professors of Computer Science
```

See Appendix B for a complete listing of XLSI's defined resources.

To reference a book title within the internal LSI structures, XLSI utilizes a user configuration file. This configuration file can be customized by modifying the user-defined variable `doc_dir` in the `storage.h` include file (and re-compiling XLSI), or by the user-defined environment variable `BOOK_CONF`. See Appendix 5.2 for a complete listing of user configurable parameters.

## 3.2 XLSI Operating Modes

XLSI can be configured in two distinct operating modes, *Search Mode* and *Browse Mode*. When operating in *Search Mode*, the user inputs are defined as constructing a search. In particular, any reference to a keyword or document signifies to XLSI that the user would like those keywords or documents added to the search criteria. If XLSI is operating in *Browse Mode*, inputs are processed as if the user is *browsing* the books and therefore, the associated text is displayed.

## 3.3 Query Construction

After a book has been selected, XLSI displays the *Control Interface*. This is the main interactive interface for the XLSI application. In this interface, the user is able to input natural language queries (search strings) for textual retrieval. This input occurs in Motif's text widget class, `XmText`. The field *Keywords:*, is used for search string input. The `XmText` widget provides the user with a single line entry field. Entry into this field can be via the keyboard or by *cutting and pasting* which is provided by the window manager. XLSI retrieves the user supplied information via the `XmGetText` Motif function and passes the search string to internal XLSI code for parsing.

Depending upon the size of the book, searching a document requires significant processing and time. For this reason, search processing is not activated by the return key or pointer input. To perform a search, the user must explicitly utilize the Motif `PushButton` labeled *Begin Search*, which is implemented using a `PushButton` widget. The `PushButton` labeled *Begin Search* is configured with the `XmNlabelString` resource, and when the user activates the pushbutton, the callback routine `XmNactivateCallback` is executed. If XLSI is currently in *Search Mode*, a search is performed. When a search is initiated, the first step is to retrieve the particular keywords and titles (documents) specified. This is accomplished by the Motif function `XmGetText` which returns the keywords and titles (documents) from the appropriate Motif `XmText` widgets. At this point, XLSI parses the users input and builds the appropriate query string. If the user has only entered keywords (terms), then the query string is just that list of terms. If the user has selected a title, which is used as a referent to a particular document, for relevance feedback searching, then the query string is the document number associated with that particular title (document), proceeded by a +. For example, if the user entered the keywords *computer chip* and selected the fifteenth document represented by the title *Intel Computer Company*, the internal search string would be constructed as `computer chip +15`. These keywords and documents are taken together to form a *pseudo-document* which is compared with documents already indexed by the truncated SVD [DDF+90].

After a search is completed, XLSI returns a list of document title numbers in rank-order (highest cosine match first). By mapping the document title to the document number, XLSI is able to retrieve the matching document text. For output display, XLSI then attaches the rank-ordered (cosine) matching information to the titles which are then placed in the *Title Interface* interface. The *Title Interface* interface uses a Motif `XmScrolledList` widget to permit the user to choose a title by *pointing-and-clicking*. XLSI uses the `XmNsingleSelectionCallback` resource to determine which title was selected. By setting the `XmScrolledList` resource, `XmNscrollBarDisplayPolicy`, to `XmAS_NEEDED`, the interface can place multiple titles into a small interface. Additionally, because this `XmScrolledList` is parented by a `Form` widget, the user can re-size the title interface to better suit their query needs.

After a successful search and after the titles are displayed in the title interface, XLSI displays the text of the first document in the rank-ordered list. This document is displayed in a separate interface using the `XmText` widget. XLSI uses the `ScrolledText` resource set of the `XmText` widget so that the user can customize the amount of text that should be displayed. If the documents in the book are relatively large, the *Document Interface's* textual output can be expanded. Likewise, if the document is small, the *Document Interface's* output size can be reduced. By using the `ScrolledText` resource, XLSI can automatically add a scrollbar when the document exceeds the size of the document text interface. The keywords that were contained in the user's search string are highlighted in the document text by using the `XmHighLightText` Motif library function.

### 3.4 Relevance Feedback

Relevance Feedback, as discussed in [SB90] and briefly described in Section 1.4 , can be a very effective search strategy. In XLSI, relevance feedback queries are formed using the entire text of any documents specified by the user. To implement relevance feedback, XLSI allows users to enter titles of documents from the *Title Interface*. To construct a query using relevance feedback, the user first clicks on the pushbutton to place XLSI in *Search Mode*. From this point, until the user toggles the pushbutton to *Browse Mode*, any click on a title places the title text in the *Title text* field of the *Control Interface*. Like the *Keyword text* field, the *Title text* field is implemented with a `XmText` widget. However, since the title must match exactly with the titles in the book, the resource `XmNeditable` is set to false. This match must occur, because to perform relevance feedback, the entire document text must be passed to the LSI search engine. To map the title to the document text, XLSI matches the titles entered on the *Control Interface* to the document number, and then XLSI retrieves the corresponding document text from the document number. This action is reflected to the user by placing the document number in the status interface located above the *Mode* button in the *Control Interface*. After the user has constructed the search with any combination of keywords and document titles, the *Begin Search* pushbutton is activated.

## 3.5 Search History

The original MGR-based interface for LSI [DS91] provided a search history interface which displayed the previous keyword searches that had been performed. The original XTDRS/XLSI [BAM93] interface provided the same history functionality with several enhancements. The third-generation interface XLSI, also implements the *Search History* interface with additional enhancements.

One of the early enhancements was to provide the users the ability to visualize their previous searches. Since XTDRS/XLSI allowed the users to search with any combination of keywords and documents (relevance feedback), a display mechanism was in place to distinguish between these two types of inputs. XLSI distinguishes between keywords and documents with the use of three visuals. First, the keywords and document titles are placed on separate lines in the *History Interface*. Secondly, the keywords are prefixed by a *K:* and the document titles are prefixed by a *T:*. Thirdly, all words in the document titles are displayed with an italic font.

By definition of LSI, words that exist in two or more documents and are not considered *common* words (*of, the, this, etc.*) are keywords. The *History Interface* provides a visual mechanism to inform users which terms (keywords) that they entered into the keyword search field were actually keywords in the XLSI search. Words or terms that were actual keywords (as defined by LSI) are displayed in italics, while other words or terms are displayed in the default font.

To implement the *History Interface*, XLSI also uses a `XmText` widget. This `XmText` widget provides the same scrolling features of the *Title Interface*. The main use of the *History Interface* is to allow the user to reconstruct a previous search. Therefore, the keywords and document titles must be easily selected. When XLSI is configured in *Search Mode*, the user simply points and clicks on the word or document title that they would like to utilize in the search. The keywords are retrieved from the *History Interface* by examining the internal Motif communication structure (*Callback Structure*). By mapping the position of the user's click (*item_position*) with the internal search history structure, XLSI is able to reconstruct the actual words and titles (documents) from the previous query. These words and titles are placed in the appropriate *Keywords:* or *Title:* fields of the *Control Interface*.

## 3.6 Previous Search Status

Although originally designed as a document retrieval tool, XLSI can be used to monitor the LSI retrieval technology. In particular, users need to understand both current and previous queries and the results returned. To address this need, XLSI provides a *Previous Search Status Interface*. Like the *History Interface*, the *Previous Search Status Interface* displays keywords and document titles used in a particular search. Additionally, the *Previous Search Status Interface* displays the actual rank-ordered listing of the documents retrieved by that search. This enhancement allows users to *track* a particular document and determine which query returned that document in the highest rank-ordering. Viewing a document title, the search criteria selected and the rank-ordering, the user has the information necessary to better interpret LSI performance and improve searching.

Implementation of the *Previous Search Status Interface* took into consideration the times at which a user might want to see previous search status information. The current design allows the interface to be displayed from the *History Interface*. Therefore, if XLSI is configured in *Browse Mode* and the user clicks on a previous search in the *History Interface*, the *Previous Search Status Interface* will be displayed. The *Previous Search Status Interface* displays the search criteria used in that particular search (terms (keywords), titles (documents), or any combination of terms (keywords) and titles (documents)) and the rank-ordered listing of the documents returned by that search. The utilization of XLSI in this capacity allows the user to click on one search, learn from the results, click on another search, compare the results, etc.

The *Previous Search Status Interface* is also implemented using a `XmText` widget. As with all other text interfaces, this `XmText` widget provides scrolling functionality. If the XLSI user only requires a small amount of document titles to be displayed, the interface's output can be resized to accommodate this display. If the user requires a large amount of document titles, the interface's output can be enlarged. Like the other interfaces, Motif's `XmNScrollBarDisplayPolicy` is configured to `XmAS_NEEDED` which displays the scroll bars when necessary. Also, to provide consistency, if XLSI is in *Search Mode*, and the user clicks on a document title, the document title text is displayed in the *Control Interface* to be used by the next XLSI search. If XLSI is configured in *Browse Mode* and the user clicks on a document title, the document text is displayed in the *Document Text Interface*.

## 3.7 Electronic Mail Capability

As XLSI evolved, the need to communicate between users grew. The knowledge of LSI configuration, retrieval results, and search criteria needs to be shared between users and LSI developers. To provide this feature, XLSI incorporates the use of electronic mail. Users can electronically mail their search results from any of the multiple interfaces within XLSI. They can email the contents of the current *Title Interface*, the *History Interface*, the *Document Text Interface*, or the *Previous Search Status Interface* to any valid internet address. In addition to electronically mailing the Interface information, XLSI allows the user to edit and annotate the information. This feature can be used to facilitate communication between XLSI users and LSI developers.

To implement the electronic mail functionality in XLSI, a pull-down menu was added to each interface (*Title, History, Document, and Previous Search Status*). The user pulls down the *File Menu* and selects *Email*. To enter the destination address, XLSI uses a `Dialog` widget and prompts the user to *Enter the e-mail address to send information*. After the user enters the address, they have the option of sending the electronic mail (*OK Button*) or canceling the operation (*Cancel Button*). XLSI implements a callback with the resource `XmNokCallback` in order to retrieve the electronic mail address. After the electronic mail address has been retrieved, XLSI executes the editing command specified by the `EDIT_COMMAND` environment variable. A sample setting for this variable might be `/usr/bin/X11/xterm -e vi`. This would pop-up a new `xterm` and use the `vi` editor to edit the information. After the information has been edited and saved, XLSI executes the user configured email command (e.g., `mail`). This immediately electronically mails the interface information along with all edits the user may have added.

## 3.8 Print Capability

After retrieving relevant information, XLSI provides the user the ability to print document text. By simply pulling down the *File* menu on the *Document Interface*, the user can select the *Print* option. This will print the document text to the user configured printer, using the user-supplied print command. XLSI provides the print feature to all interfaces (*Title Interface, History Interface, Previous Search Status Interface and Document Text Interface*).

To provide the maximum level of customization and to allow XLSI to work in a heterogeneous computing environment, the print command is customizable. By either modifying the `lp_command` in the `storage.h` include file, or by setting the `LP_COMMAND` environment variable, a user is able to execute the print command specific for their computing environment, as well as specify their printer destination and any unique print options. See Appendix 5.2 for information concerning user customizable environment variables.

## 3.9 Trace Files

XLSI implements a *Save* feature for all the interfaces (*Title Interface, History Interface, Previous Search Status Interface and Document Text Interface*) which allows a user to trace their XLSI session. This provides an *audit* trail of the searches and results for later analysis of LSI performance. To save information, the user selects the *Save* option from the *File* menu. The interface information is saved to a user-configured save file.

## 3.10 LSI Options

As documented in Section 1.2, the user may desire *similar* documents to be ranked (or clustered) together by reducing the number of LSI factors or dimensions that were generated by the SVD. XLSI provides an interface that will allow the user to increase or decrease the number of factors (dimensions) to be used in the next search. This functionality allows the experienced LSI user to perform a query $x$ with a factor of $y$, and then reconfigure the LSI search to perform query $x + 1$ with $z$ factors. However, the user cannot increase the number of factors to be greater than the original number of factors generated by the original truncated SVD. To modify the number of factors, the user selects *LSI Options* from the *Control Interface's* menu-bar. When the pulldown menu option *LSI Configuration* is selected, the *LSI_Window* is displayed. At this point, the user can view the LSI Configuration parameters including, *number of factors, number of documents, and number of terms.* The input cursor is placed in the `XmText` field associated with the *number of factors.* To modify the *number of factors*, the user enters the new number and selects *Save Configuration and Exit* from the *File* pulldown menu. If the user attempts to increase the value of factors beyond the default, an `Error Dialog` widget is displayed and the error, *Cannot increase the number of factors greater than the default value determined by LSI* is displayed.

Additional LSI information displayed by the *LSI Options Interface* is the number of documents in the selected book and the number of of terms (words) used by LSI in searching the book. The default number of documents returned after a query is 50 or less (less if the book contains less than 50 documents). This can be modified to suit a particular search requirement. By selecting the *Change Number of Titles Displayed Interface*, the user simply supplies the number of documents that they would like returned. This allows the user to eliminate the display of irrelevant (low matching) documents or expand the number when they are experiencing problems finding their information.

## 3.11 XLSI Help

XLSI was designed to be used with a minimal amount of documentation. However, there are some interactions and information that may require additional explanation. Rather than expect the user to maintain XLSI documentation, we designed an on-line help facility. Each *Help* submenu provides an explanation of the particular interface's displayed information and the interaction of the interface's interface. The interfaces included in the help subsystem are *Control Interface, Title Interface, History Interface, Document Interface and Previous Search Status Interface.*

On-line help is implemented using a new feature of Motif 1.2, *tear-off* menus. When the user activates the help pull-down menu, a sub-menu becomes available. This submenu is created as a unique interface and can be manipulated separately. Therefore, when the user asks XLSI for help, the help menu becomes available throughout the application. The user simply places the *Help Tear-off* submenu on their interface and the particular sub-menus can be activated at any time. The tear-off menu is implemented by setting the resources `XmNtearOffModel` and `XmTEAR_OFF_ENABLED`. And the Help menu pane is placed in the right most location on the *Control Interface* by setting the resource *XmNmenuHelpWidget*.

## 3.12 XLSI Activity Alerts

XLSI's predecessor, MGR was prototyped and tested using small on-line databases or books. However, XLSI was designed for searching large on-line databases such as a condensed encyclopedia (Columbia Condensed Encyclopedia), a collection of several months of local newspaper articles (Knoxville News-Paper) and several translations of the Gospels of the Bible. These large databases motivated both efficient search algorithms and a mechanism for informing the user that work was being performed. The first release of XTDRS/XLSI provided a status interface that displayed the number of documents searched (in increments of 100). However, this in itself required processing and update time. The latest generation of XLSI now simply *locks* the cursor by changing its display icon to be a *stopwatch* and *freezing* interactivity to the application. The user is provided with a visual (a new cursor icon) to inform them that background processing is occurring. When the entire book has been searched, the cursor returns to its normal mode and the interfaces are ready to receive input.

## 3.13 Opening and Closing Different Books

When implemented in the John C. Hodges Library on the campus of the University of Tennessee, XTDRS/XLSI [BAM93] provided an interface to one book at a time. To close that book and open another book required the user to exit the application. As an enhancement to this, XLSI provides the capability to open a new book. While on the surface this appears trivial, the termination of the underlying data structures and the reinitialization of global flags have to occur. To close the current book and open a different book, the user simply chooses *Open New Book* from the *Control Interface*.

# Chapter 4

# Sample XLSI Session

In this chapter, the features and functionality of XLSI are demonstrated. The session begins with book selection followed by a typical *keyword* query. That query is then followed by a relevance feedback query using the text of a selected document. Throughout the session, the advanced enhancements of XLSI are demonstrated.

## 4.1 Customizing XLSI

The customization of XLSI is consistent with other X-Windows applications. In particular, a resource file (*xlsi.rf*) is used to notify X-Windows of default values for particular resources (e.g., Background Color). For simplicity, the notification to the X-Windows manager is provided through the local XENVIRONMENT variable. For this session, the user executes,

```
XENVIRONMENT=xlsi.rf
export XENVIRONMENT
```

## 4.2 Sample Book Selection

In XLSI, searches are performed on a collection of documents, or books. The book selection is illustrated in Figure 4.1. In this example, `UTK's Computer Science Department's Staff` was selected. To select this particular book, the user places the pointer (in this case, the mouse) over the book title and double clicks. XLSI retrieves the selection, compares `UTK's Computer Science Department's Staff` to the book titles documented in *BOOK_DIR* and locates the underlying LSI database files. If any LSI or XLSI files are missing, XLSI will report an error.

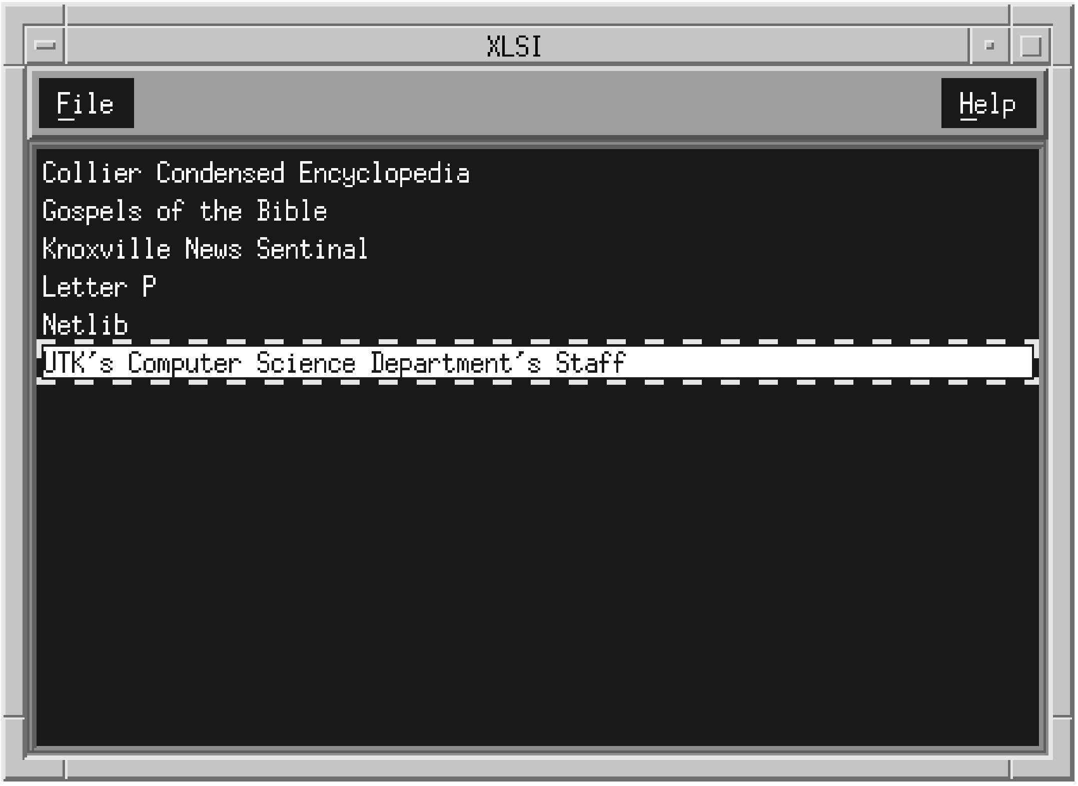



Figure 4.1: Book Selection Interface

## 4.3 Query Construction

After selecting the University of Tennessee's Computer Science Department's Staff, the user obtains the XLSI *Control Interface, the Title Interface, the History Interface, and the Document Text Interface.* The environment is illustrated in Figure 4.2. The *Title Interface* outputs the initial book titles as they appear in the book selected. In this example, each professor's name and title are displayed. The *Document Interface* initially displays the textual information associated with the first professor in the actual book text. Because this is the first search, the *History Interface* does not contain any prior searches.

### 4.3.1 Sample Keyword Search

Suppose a user is trying to find professors that conduct research in parallel computing. To begin the search, the query `research in parallel computing` can be entered. First, the *Keywords text* field is activated with the mouse (the outline of the text field will highlight upon activation). Because this text field is editable, any input errors that may occur can be corrected with the backspace key, cut and paste, or any other standard editing functions.

To indicate to XLSI that a search is to be performed, the *Mode button* is toggled with the left mouse. The *Mode button* then displays *Search Mode*. To initiate a search, the *Begin Search button*, located above the *Mode button* on the *Control Interface* is selected. As the search is being performed, the cursor is displayed as an stopwatch and user input is frozen. When the search completes, the cursor is returned to its normal icon and input is allowed in the interface.

If any words that the user entered are indexed terms (keywords) in the underlying LSI database the search can proceed. Otherwise if no words were indexed, an error message is reported and the user can enter another search. Following a successful search, XLSI reports the *matching* terms (words) in the *Status Field* located above the *Mode button* in the *Control Interface*. In this example, the keywords `research parallel computing` were valid terms and therefore utilized in the search. The *Title Interface*, as displayed in Figure 4.3, contains a rank-ordered list of the returned documents with each title preceded by its relevance factor (100 times cosine with query). The initial XLSI configuration displays a maximum of 50 documents (or all the documents if the total is less than 50). The vertical scrollbar on this interface can be used to see the rank-ordering of all the professors as judged by LSI. The horizontal scroll bar can be used to fully display the professor's names and titles. The *History Interface* (Figure 4.4) logs the queries and *italicizes* the keywords found and **bolds** the words that were indexed by XLSI. Queries based solely on keywords are preceded by a *K:* in the *History Interface*.

Finally, the *Document Interface* is updated to contain the text of the top-ranking document. In this example, the highest matching professor is *Michael A. Langston; Professor of Computer Science*. Figure 4.5 contains the text window that displays the information associated with this professor. The scroll bar in the *Title Interface* can be used to move down the rank-ordered list of professors. When the next professor is highlighted in the *Title Interface*, their associated text is displayed in the *Document Interface*. Any keyword from our most recent query (if the query was a keyword search) is highlighted in the *Document Interface*. This highlighting remains in effect when browsing articles returned.

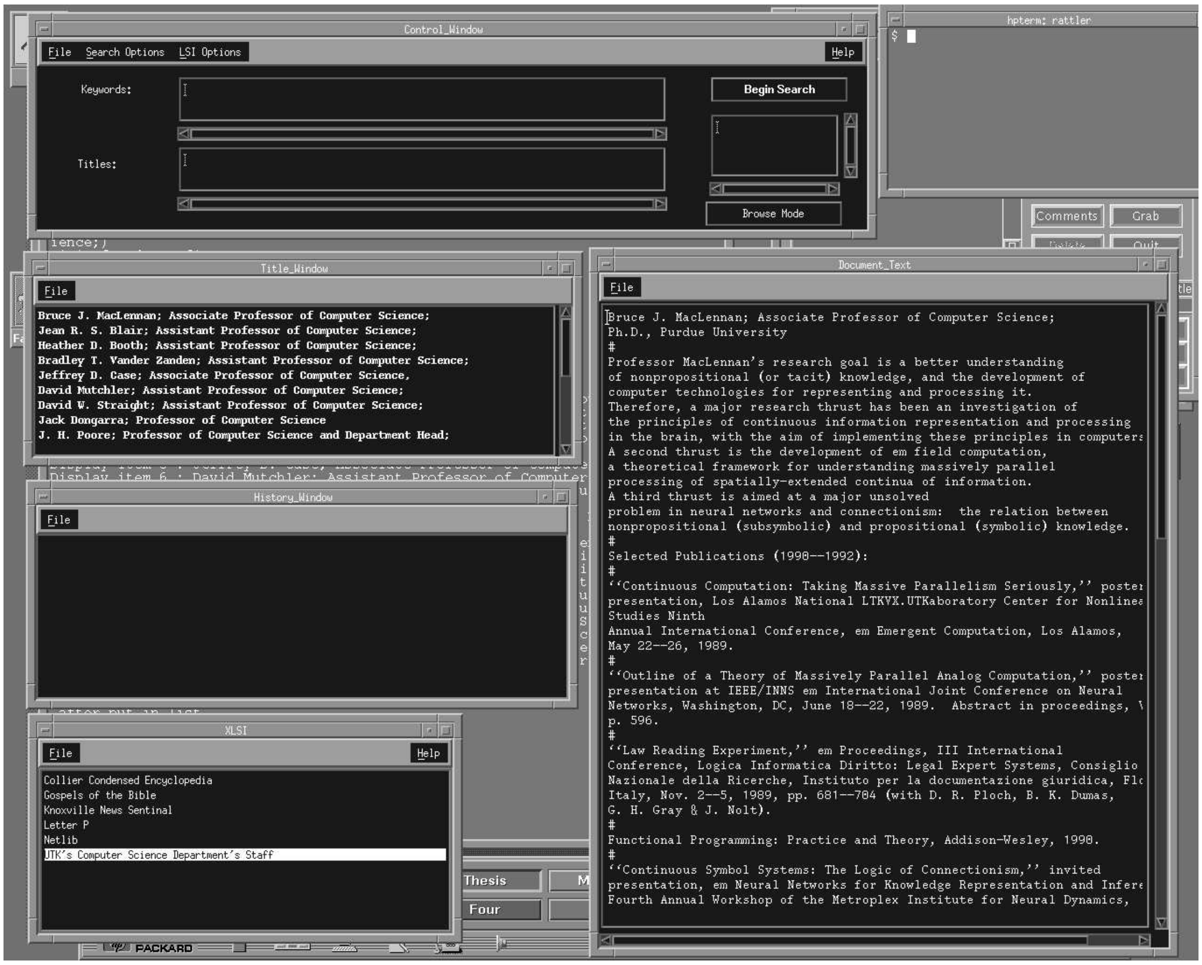



Figure 4.2: XLSI Interactive Environment

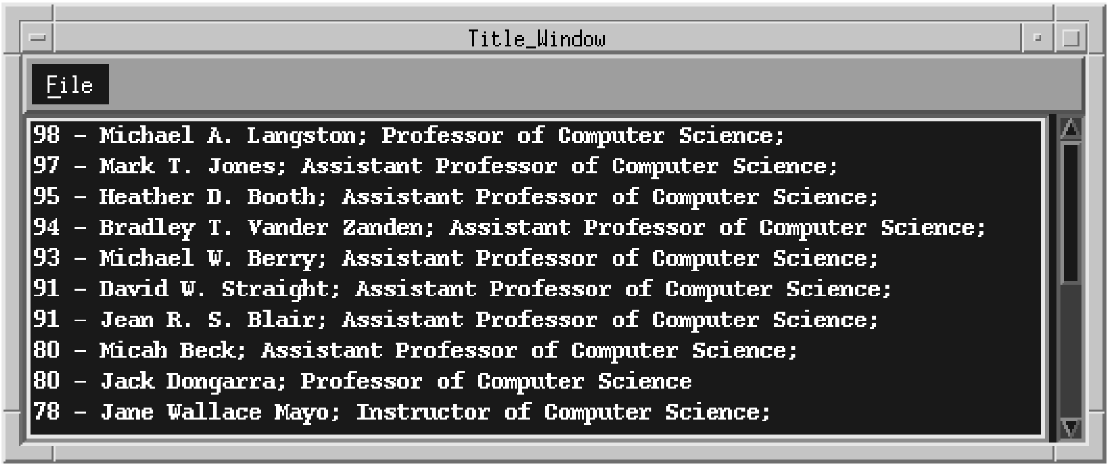



Figure 4.3: Title Interface

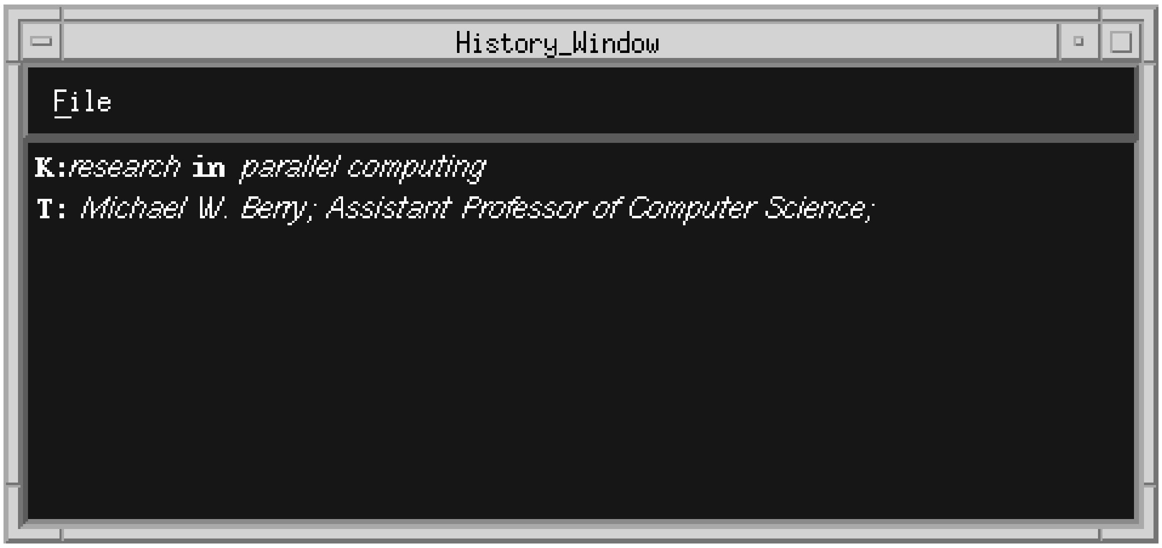



Figure 4.4: History Interface

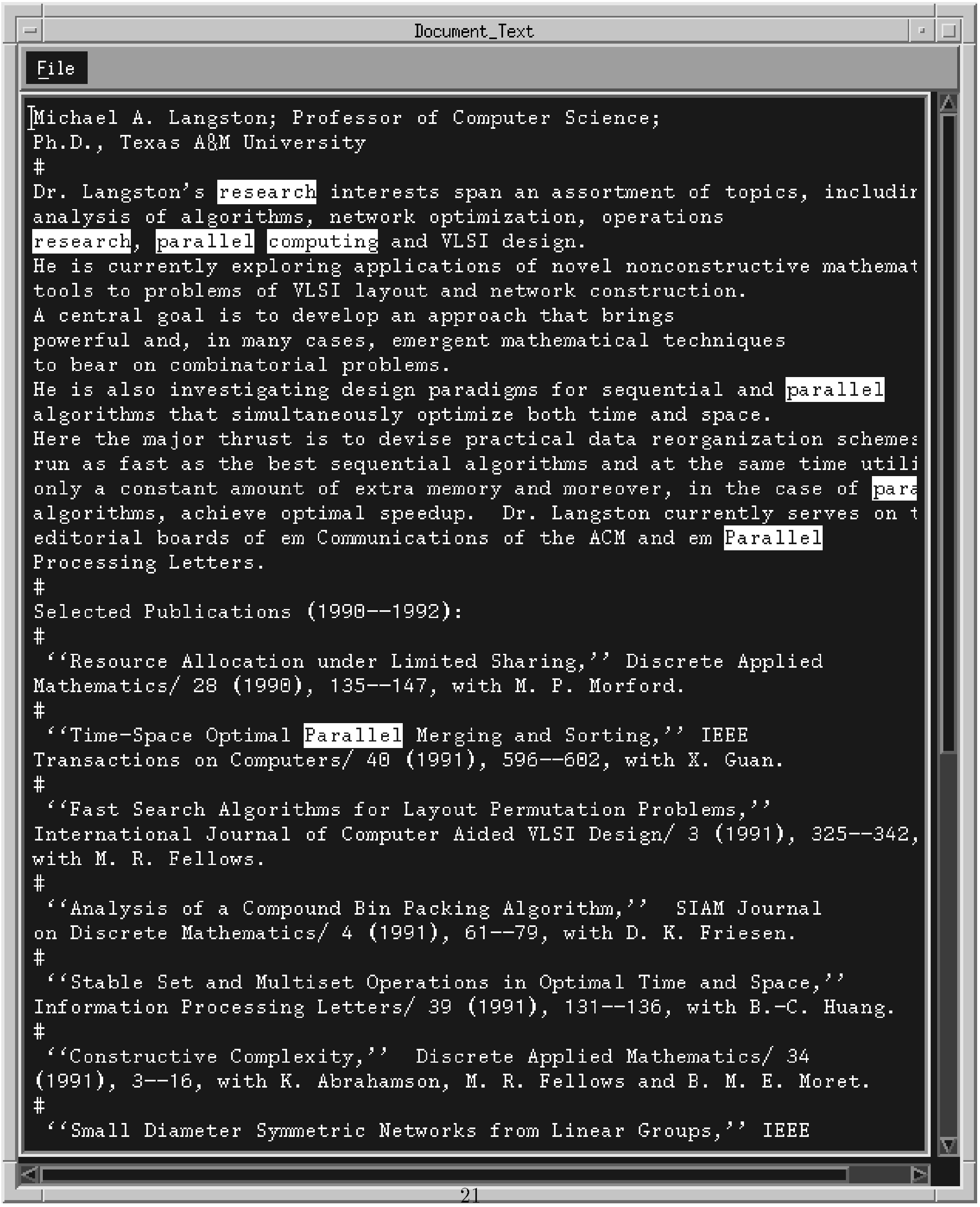





Figure 4.5: Document Interface

### 4.3.2 Sample Relevance Feedback Search

As described in Section 3.4, XLSI allows users to find *like* documents using relevance feedback. In essence, the entire document is used as a search, in hopes of retrieving similar documents. This feature is implemented in XLSI by allowing the user to select any number of titles (used as referents to documents) from the *Title Interface*. The query is then built with the entire document text associated with each title entered in the query.

After scrolling the *Title Interface*, the text associated with *Michael W. Berry* is displayed (see Figure 4.6). Suppose the user wants to find the professors of computer science who's research is most like Dr. Berry's. After toggling the *Mode button* to *Search Mode*. the user selects the title `Michael W. Berry; Assistant Professor of Computer Science` from the *Title Interface*. Because XLSI is now in *Search Mode*, a query is constructed. The title selected is then displayed in the *Titles:* field of the *Control Interface*. Dr. Berry's text is simultaneously displayed in the *Document Interface*. To initiate the search, the *Begin Search* button is pressed and Figure 4.7 shows the result of this relevance feedback search. The *Title Interface* has been updated to reflect the new search, the *Document Interface* contains the text associated with the highest-ranking document and the *History Interface* displays the query. Since this query was constructed with a document rather than a keyword, the *History Interface* proceeds the title with a *T:* and *italicizes* the entire title. Notice that the article on Dr. Berry is the highest-ranked document with a relevance factor of 100 (i.e., the returned document is identical to the search document or query). To indicate that a relevance feedback search was performed, the *Control Interface* displays the actual document number (+15) in the *status field*.

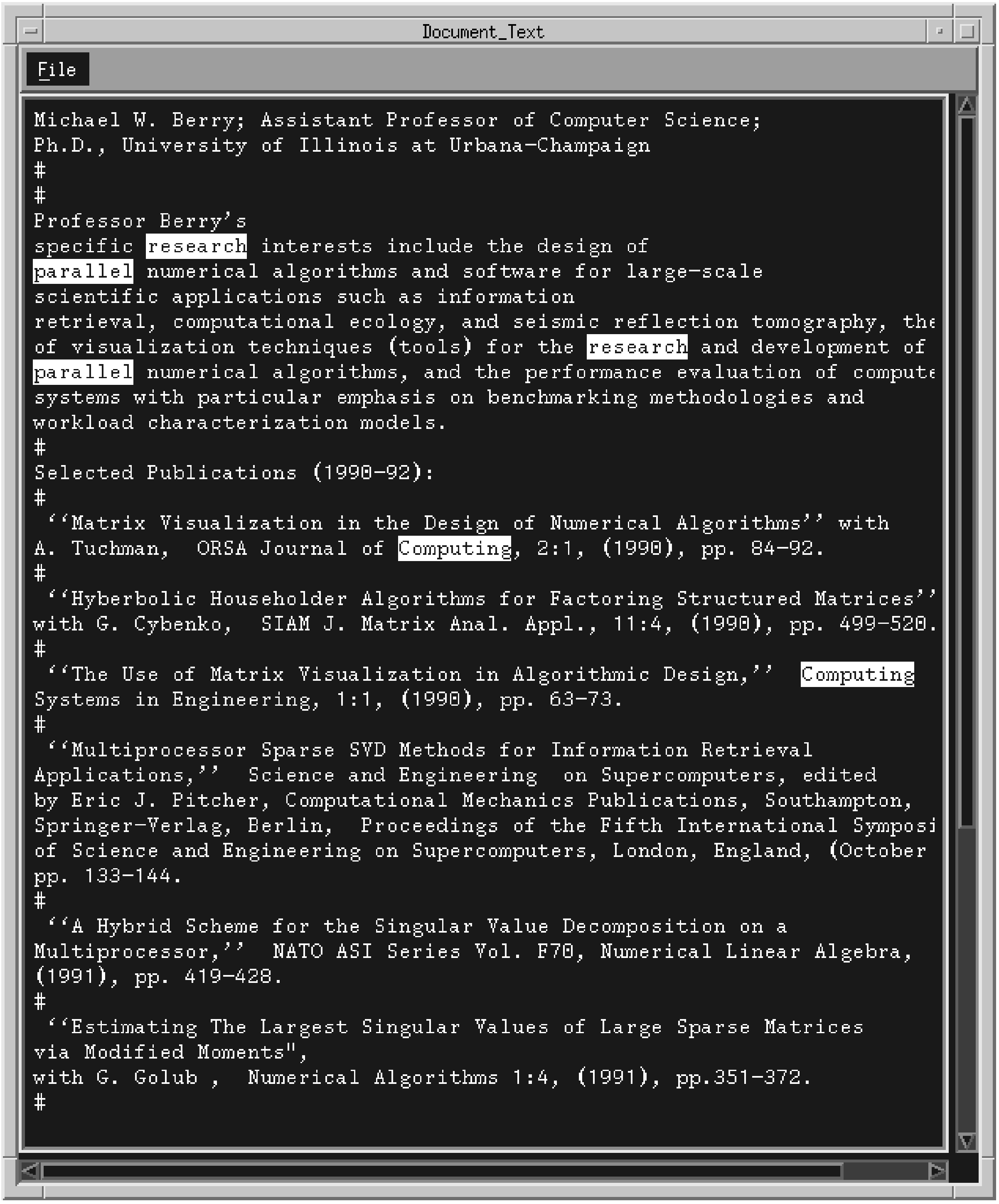





Figure 4.6: Document Interface containing text on Dr. Michael Berry

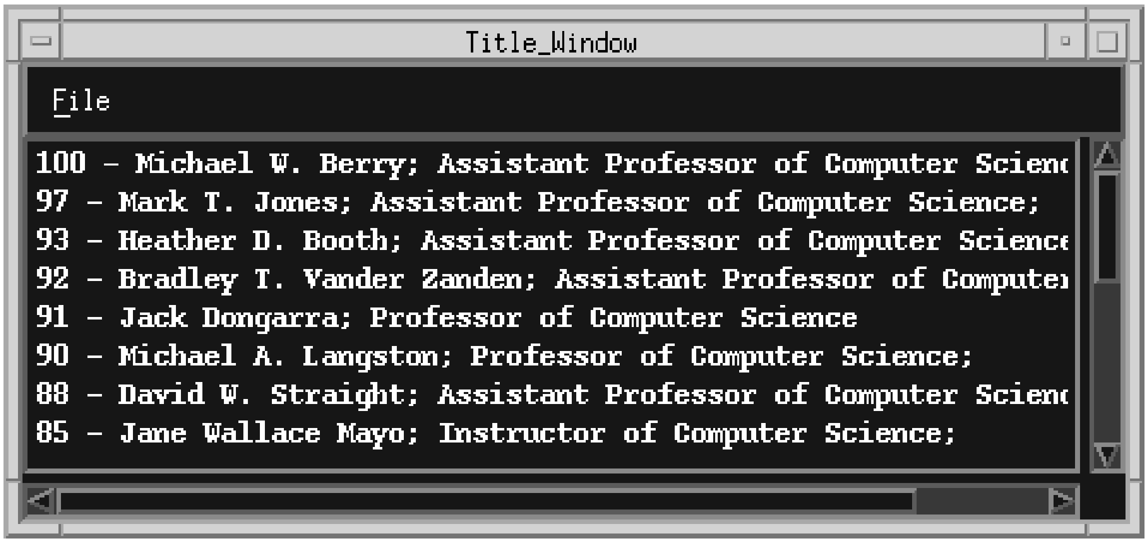



Figure 4.7: Title Interface after Relevance Feedback Search

## 4.4 Utilization of Search History

The original MGR-based interface for LSI provided a search history interface which displayed the keyword queries that had been performed. Because XLSI provides the capability of performing searches with keywords, titles (documents), or keywords and titles (documents), a method to inform the user of the types of search previously performed was needed. To distinguish the different types of searches several display features have been incorporated. First, the keywords and (titles) documents are reported on a separate line in the *Search History Interface*. Second, the keywords are preceded by a *K:* and the titles are preceded by a *T:*. Third, as illustrated in Figure 4.4, titles are displayed in *italic*.

Another function of the *Search History Interface* is to allow the user to reconstruct a previous search. XLSI provides this functionality by utilizing a *point-and-click* interaction with the interface. To reconstruct a query, the user puts XLSI in *Search mode* by toggling the *Mode button*, and then selects any combination of the previous searches (keywords, titles (documents), or keywords and titles (documents)). XLSI determines if the previous search selected was a keyword or title (document) and fills the appropriate text field in the *Control Interface*. After the user has constructed their new query, they press the *Begin Search button* to perform the search. Figure 4.8 illustrates the selection of a new query based on two prior searches, the *Keyword* search `research in parallel computing` and the *Title* (Document) search `Michael W. Berry, Associate Professor of Computer Science.`

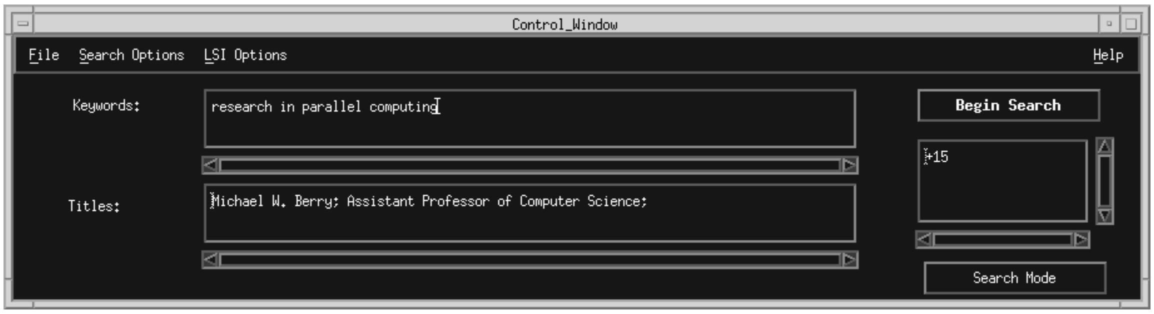



Figure 4.8: Search as reconstructed using the History Interface

## 4.5 Building a Previous Search

With the original MGR based-interface, and XTDRS/XLSI, there was no *memory* or *display* of results from previous searches. XLSI provides previous search results in a separate interface, *Previous Search Status Interface*. The information displayed in the *Previous Search Status Interface* is the actual search strings (keywords, titles (documents), or keywords and titles (documents)) and the resulting rank-ordered listing of the titles (documents).

To display the previous search information, the user places XLSI in *Browse Mode* by toggling the *Mode button* on the *Control Interface* and then clicks on the desired previous search from the *History Interface*. XLSI searches the internal data structures to find the previous search information associated with the selected search and displays the information, see Figure 4.9. In displaying the previous search criteria, XLSI displays titles (reflecting relevance feedback searches) and keywords on consecutive lines in the *Previous Search Interface*.

With large databases, long searches, and consecutive searches, the amount of data managed by XLSI can be enormous. To avoid memory limitations, XLSI stores the results of only the ten most recent previous searchs. When the user performs the eleventh consecutive search, XLSI informs them that the ten latest searches are stored and that some previous search information will be discarded. Also, if the user tries to retrieve search status information that has been overwritten in memory, XLSI reports an error message.

## 4.6 LSI Configuration

As mentioned in Section 1.5 the user may attempt to map *like* documents closer by reducing the number of factors or dimensions. To modify the number of dimensions to be evaluated, the user picks the *LSI Options* menu item from the *Control Interface*. The default number of factors, number of documents and number of terms is displayed (see Figure 4.10). To modify the number of factors, the user places the pointer in the text field to the right of the *Number of factors* label and enters the new value. To activate these changes, the user selects the *Save Changes and Exit* menu selection from the *File* menu-bar pane. If the user attempts to increase the number of factors beyond the original LSI, an error dialog is displayed.

## 4.7 Electronic Mail Communication

The original MGR-based retrieval tool and XTDRS/XLSI provided information to a single user. XLSI extends this communication to allow information to be passed between multiple users. This is accomplished by incorporating electronic mail. Lit-

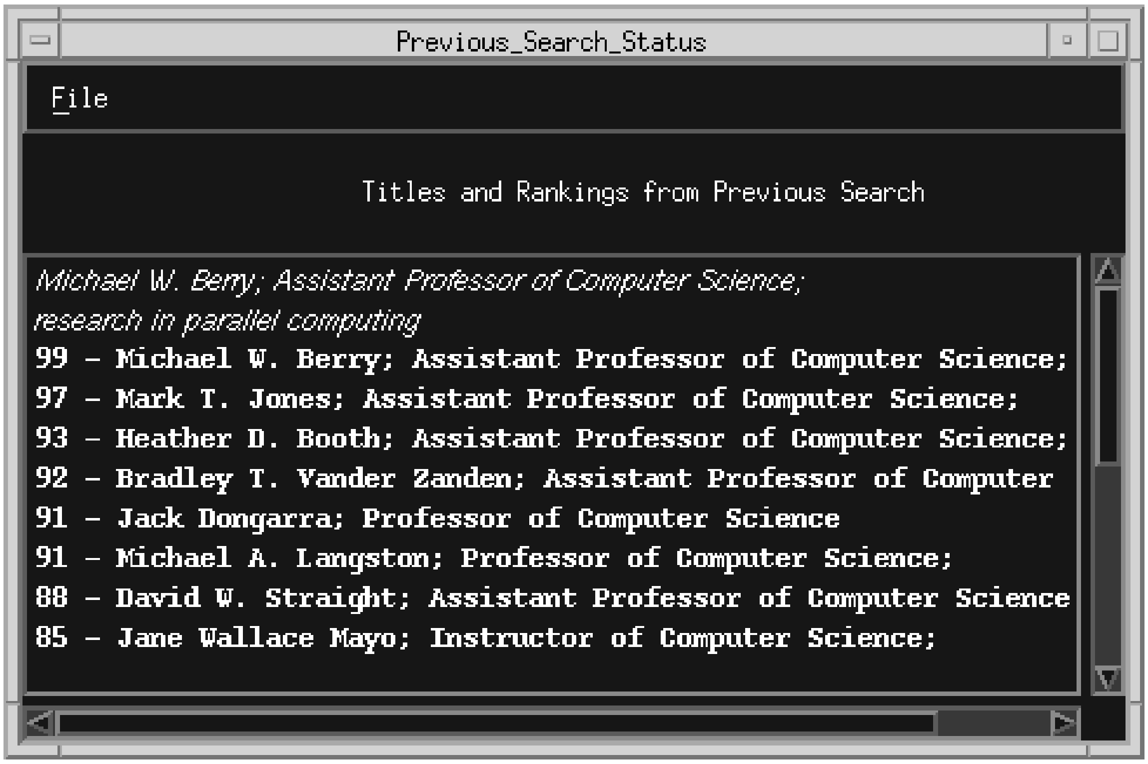



Figure 4.9: Previous Search Status Interface

erally, any information displayed by XLSI can be electronically mailed to another user. And to enhance communications, the user can annotate the information prior to mailing it to another user.

For example, suppose the user wanted to send electronic mail about the rank-ordered listing returned in the XLSI search mentioned above. After activating the *Title Interface* containing the rank-ordered information and pulling down the *File menu* they may select the *E-Mail* option. XLSI then prompts the user for the electronic mail address as displayed in Figure 4.11. After the appropriate information has been provided, the user clicks the *OK button*. At this point, XLSI opens an edit interface (see Figure 4.12) with the recorded information from the *Title Interface*. After the user edits the information and closes their chosen editor, the modified information is mailed. This process can be repeated indefinitely and with any information displayed by XLSI.

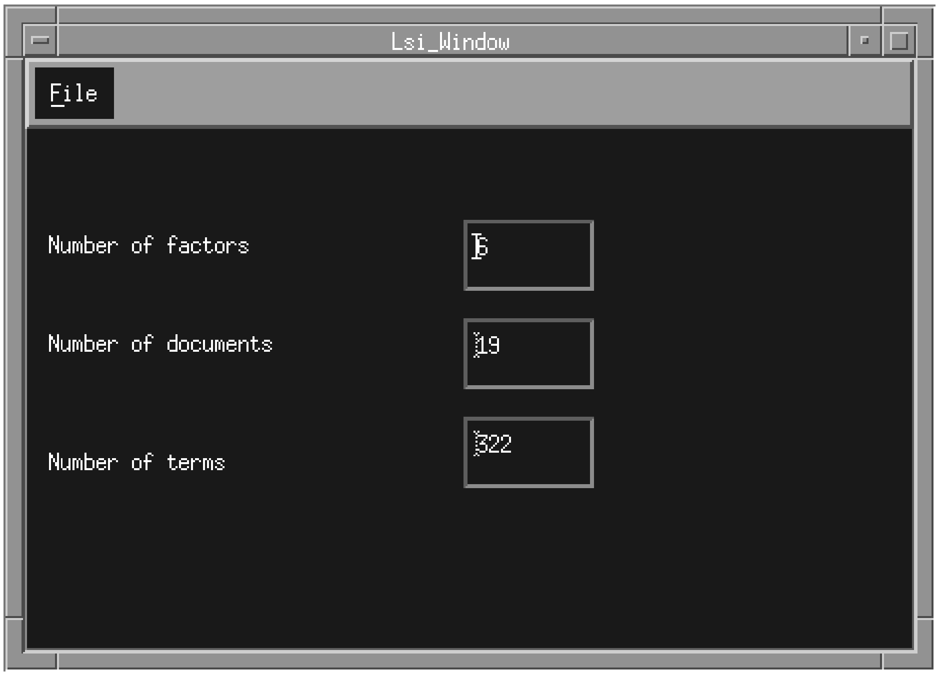



Figure 4.10: LSI Options Interface

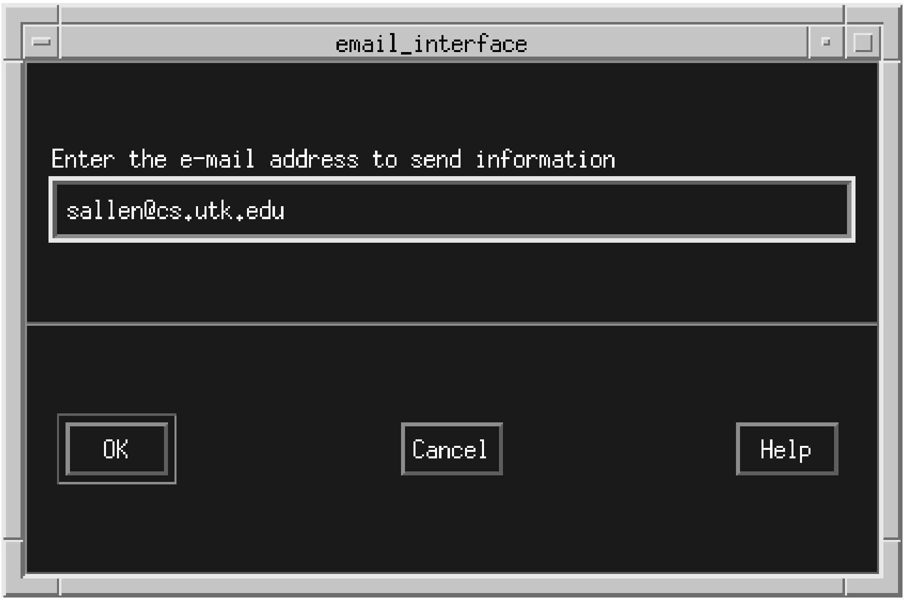
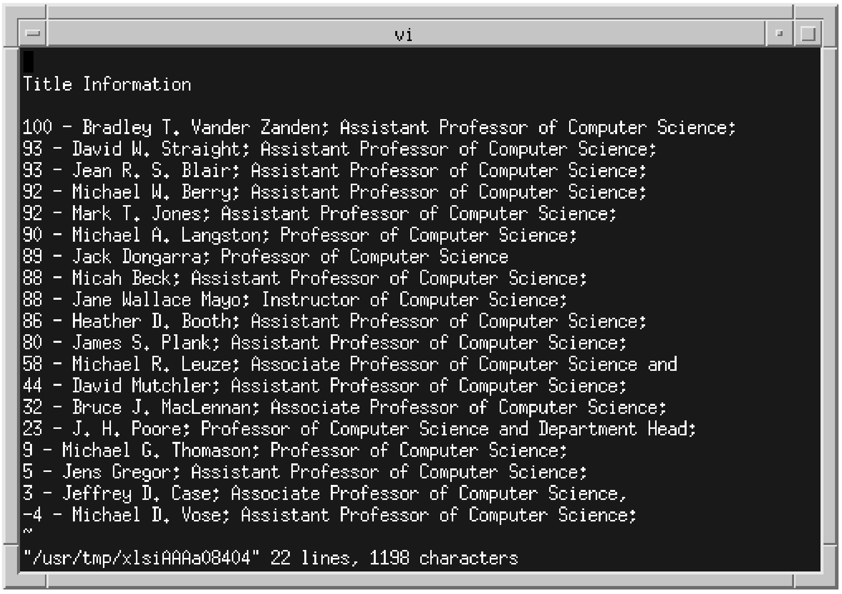



Figure 4.11: Electronic Mail Address Interface



```
vi

Title Information

100 - Bradley T. Vander Zanden; Assistant Professor of Computer Science;
93 - David W. Straight; Assistant Professor of Computer Science;
93 - Jean R. S. Blair; Assistant Professor of Computer Science;
92 - Michael W. Berry; Assistant Professor of Computer Science;
92 - Mark T. Jones; Assistant Professor of Computer Science;
90 - Michael A. Langston; Professor of Computer Science;
89 - Jack Dongarra; Professor of Computer Science
88 - Micah Beck; Assistant Professor of Computer Science;
88 - Jane Wallace Mayo; Instructor of Computer Science;
86 - Heather D. Booth; Assistant Professor of Computer Science;
80 - James S. Plank; Assistant Professor of Computer Science;
58 - Michael R. Leuze; Associate Professor of Computer Science and
44 - David Mutchler; Assistant Professor of Computer Science;
32 - Bruce J. MacLennan; Associate Professor of Computer Science;
23 - J. H. Poore; Professor of Computer Science and Department Head;
9 - Michael G. Thomason; Professor of Computer Science;
5 - Jens Gregor; Assistant Professor of Computer Science;
3 - Jeffrey D. Case; Associate Professor of Computer Science,
-4 - Michael D. Vose; Assistant Professor of Computer Science;
~
"/usr/tmp/xlsiAAAa08404" 22 lines, 1198 characters
```

Figure 4.12: Electronic Mail Editing Interface

## 4.8 XLSI Printing Capabilities

One enhancement of XLSI is the ability to print any information displayed or retrieved. The original version of XTDRS/XLSI provided the ability to print only the document text (via the *Document Interface*). However, to provide additional functionality for those users attempting to learn more about LSI, XLSI provides the capability to print any information retrieved. The user simply selects the *File menu* on any screen, then selects the *Print option.* This will output the information to the user configured printer. See Appendix 5.2 concerning the details of customizing the print environment.

## 4.9 Obtaining on-line help

To retrieve help throughout XLSI, the user may select the *Help button* on the *Control Interface*. The *Help sub-menu* is displayed and the user can retrieve more detailed information by selecting a particular help topic. The *Help sub-menu* was designed and implemented utilizing Motif 1.2's tear-off menus. This allows one help screen to be available throughout the entire application. To display on-line help to the entire application, the user *tears-off* the help menu and locates it anywhere on the screen (see Figure 4.13).


Figure 4.13: Help Tear-off Menu

# Chapter 5

# Current Platforms and Future Work

## 5.1 Current Platforms

One of the initial design goals for XLSI was portability across many computing platforms. After considering several commercially-available windowing toolkits, OSF's (Open Software Foundation) Motif was chosen. The OSF/Motif toolkit is available on most hardware platforms, including Hewlett-Packard, IBM, DEC, SUN and SGI. This allowed development on a single platform, yet cross-compile onto the platform required by users.

XLSI was developed on a Hewlett-Packard Series 9000 Model 720 computer at The University of Tennessee at Knoxville. The configuration included 64MB of main memory and 600MB of disk space. The software configuration includes: HP-UX 9.01, X-Windows R5 and OSF Motif 1.2. Because LSI is highly computational in nature and XLSI manipulates large amounts of data, the high-performing HP workstation provided an excellent development platform. To provide XLSI to additional computing communities, we have also compiled and executed on a Sparc Station 10, utilizing IXI's Motif libraries.

## 5.2 Future Work

In this thesis, XLSI, a Graphical User Interface for a Conceptual Retrieval System, has been presented. As documented, the underlying retrieval technology for XLSI is LSI, which includes relevance feedback. While relevance feedback has been proven to be a very successful retrieval technology [DFL88], users of XLSI still question the results. Most of the confusion is because the actual keywords that the user entered either might not exist in the documents retrieved or that the documents retrieved might not be ranked by the total number of keywords (hits) found. One solution to this problem might be to incorporate an actual keyword search mode in XLSI. By allowing the user to toggle between keyword search mode and fuzzy (LSI) search mode, the users might gain a better understanding of relevance feedback searches.

With the growth of multimedia information and applications, one enhancement to XLSI would be to redesign the underlying information databases. The current design maintains all data structures in memory and while this allows for fast access, it does limit the amount of information that can be accessed. With an underlying database management system handling the data access and manipulation, XLSI can be expanded to handle even larger books and more diverse information.

# Bibliography

# Bibliography


[BAM93] M. Berry, S. Allen, and R. MacIntyre. XLSI: A motif-based user interface for a conceptual retrieval system. *The X Journal*, 3(2):58–64, 1993.

[Ber92] M. W. Berry. Large scale singular value computations. *International Journal of Supercomputer Applications*, 6(1):13–49, 1992.

[DDF+90] S. Deerwester, S. Dumais, G. Furnas, T. Landauer, and R. Harshman. Indexing by latent semantic analysis. *Journal of the American Society for Information Science*, 41(6):391–407, 1990.

[DFL88] S. Dumais, G. Furnas, and T. Landauer. Using latent semantic analysis to improve access to textual information. In *Proceedings of Computer Human Interaction '88*, pages 281–285, 1988.

[DS91] S. T. Dumais and D. G. Schmidt. Iterative searching in an online database. In *Proceedings of the Human Factors Society 35th Annual Meeting*, pages 398–402, 1991.

[Dum91] S. T. Dumais. Improving the retrieval of information form external sources. *Behavior Research Methods, Instruments, & Computers*, 23(2):229–236, 1991.

[SB90] G. Salton and C. Buckley. Improving retrieval performance by relevance feedback. *Journal of the American Society for Information Science*, 41(4):288–297, 1990.

# Appendices

The following environment variables have been provided to allow maximum customization of the XLSI environment. An initial configuration is provided in the `xlsi.csh.env` file. This file, `xlsi.csh.env` was constructed at The University of Tennessee, using the `csh` (c shell). `xlsi.env` is provided for the `sh` (bourne shell) and `ksh` (korn shell) users.

### BOOK_DIRECTORY

`BOOK_DIRECTORY` is an environment variable that provides the XLSI code the location (directory) of the specific books. The `BOOK_DIRECTORY` is the global directory. The sub-directories represent the specific books by name. For example, the directory structure of `BOOK_DIRECTORY` could be:

```
/straw/homes/berry/books/
/straw/homes/berry/books/CCE
/straw/homes/berry/books/CSABS
/straw/homes/berry/books/UTK-CS
/straw/homes/berry/books/KNOXNS
```

### BOOK_CONF

`BOOK_CONF` is used by XLSI to make the connection between the book that the user selected and the actual directory of SVD information. This very basic implementation is accomplished as follows:

```
UTK's                            UTK-CS
Knoxville-News-Sentinel          KNOXNS
UTKCS-Tech-Report-Abstracts      CSABS
```

When the user selects UTK's Computer Science Department Staff from the beginning XLSI interface, XLSI opens the `BOOK_CONF` file and maps the UTK's string to the UTK-CS directory. This directory name is appended to the `BOOK_DIRECTORY` to form the final search path for the LSI information:

```
/straw/homes/berry/books/UTK-CS
```

**HELP_FILE**

`HELP_FILE` points to the help text that will be displayed on the initial screen. By providing a user configurable file, the user or XLSI developer can customize the degree of help available through modifying the `HELP_FILE`. Again, this allows customization without modifying the actual XLSI source code.

**SAVE_FILE**

When the user selects the `Save` option from the `FILE` menu of any of the interfaces, the displayed information is saved to the file pointed to by `SAVE_FILE`. This allows the user to customize the location of the saved information for this execution of XLSI. New information saved is appended to the existing file.

**LP_COMMAND**

`LP_COMMAND` is the exact lp command that XLSI executes to print the interface information. This customization allows XLSI to execute on multiple environments with different system print commands. An example `LP_COMMAND` is:

```
/usr/ucb/lpr -Pbutter
```

**EDIT_COMMAND**

The `EDIT_COMMAND` is the system command that is executed when the user is emailing information to another user. As mentioned, the user is allowed to annotate the information that is mailed to the supplied user. By allowing the user to customize the `EDIT_COMMAND`, XLSI can execute on multiple system platforms. An example `EDIT_COMMAND` is:

```
/usr/bi/usr/local/X11R5/bin/xterm -e vi
```

**EMAIL_COMMAND**

`EMAIL_COMMAND` is the system command executed by the email interface. After the user supplies the destination address, the address is added to the `EMAIL_COMMAND` and the information is mailed. Again, by allowing the user to customize the `EMAIL_COMMAND`, XLSI can execute on multiple system platforms. An example `EMAIL_COMMAND` is:

```
/usr/ucb/mail
```

**XENVIRONMENT**

`XENVIRONMENT` is the environment variable used to pass resource definitions to the X-Windows environment. The appearance of XLSI can be modified via the resource variables to each of the separate interfaces. `XENVIRONMENT` is "read" by the window manager and the resource variables set within the file will be applied to the XLSI interface.

**XLSI_HELP**

`XLSI_HELP` provides the file pointer to the text used to explain the specifics about LSI text retrieval.

**CONTROL_HELP**

`CONTROL_HELP` provides the file pointer to the text used to explain both the information displayed in the *Control Interface* and the specifics about user interaction within this interface.

**FILE_HELP**

`FILE_HELP` provides the file pointer to the text used to explain both the information displayed in the *Control Interface's File Menu* option.

**SEARCH_HELP**

`SEARCH_HELP` provides the file pointer to the text used to explain both the information displayed in the *Control Interface's Search Options Menu* option.

**DISPLAY_HELP**

`DISPLAY_HELP` provides the file pointer to the text used to explain the *LSI Options Menu Option* of the *Control Interface*.

**TITLE_HELP**

`TITLE_HELP` provides the file pointer to the text used to explain both the information displayed in the *Title Interface* and the specifics about user interaction within this interface.

**HISTORY_HELP**

`HISTORY_HELP` provides the file pointer to the text used to explain both the information displayed in the *History Interface* and the specifics about user interaction within this interface.

**DOC_HELP**

`DOC_HELP` provides the file pointer to the text used to explain both the information displayed in the *Document Interface* and the specifics about user interaction within this interface.

**PREVIOUS_HELP**

`PREVIOUS_HELP` provides the file pointer to the text used to explain both the information displayed in the *Previous Interface* and the specifics about user interaction within this interface.

**ERROR_HELP**

`ERROR_HELP` provides the file pointer to the text used to explain the errors that can occur within XLSI. This text is displayed when the user encounters an error and chooses the *Help* pushbutton on the *Error Interface*.

XLSI provides two default sets of X Windows Resources. The first set of resources is either defined in the source code, or defined by the actual defaults to the widgets displayed. The second set of resources is provided in the file `xlsi.rf`. This file can be modified by the end-user to produce the interface "look and feel" that they desire. Following is a list of the defaults resources and the interfaces that they effect.

```
! Resources for the Document Text Interface

*.Document_Text_shell.x: 390
*.Document_Text_shell.y: 10
*.Document_Text_shell.width: 610
*.Document_Text_shell.height: 740
*.Document_Text.width: 610
*.Document_Text.height: 740
*.Document_Text.background: #9d94a5

*.scrolledText3.background: #dbc9c9
*.scrolledText3.fontList: -Bitstream-Prestige-Medium-R
  -Normal--16-120-72-72-M-80-HP-Roman8

! Resources for the History Interface

*.History_Window_shell.x: 10
*.History_Window_shell.y: 490
```

```
*.History_Window_shell.width: 460
*.History_Window_shell.height: 190
*.History_Window.width: 460
*.History_Window.height: 190
*.History_Window.background: #9d94a5

*.scrolledList3.background: #dbc9c9

! Resources for the LSI Options Interface

*.Lsi_Window.x: 220
*.Lsi_Window.y: 320
*.Lsi_Window.width: 450
*.Lsi_Window.height: 300
*.Lsi_Window.background: #9d94a5

! Resources for the Previous Search Status Interface

*.Previous_Search_Status.x: 230
*.Previous_Search_Status.y: 200
*.Previous_Search_Status.width: 490
*.Previous_Search_Status.height: 310
*.Previous_Search_Status.background: #9d94a5

! Resources for the Error Dialog Interface
```

```
*.errorDialog1_shell.x: 300

*.errorDialog1_shell.y: 290

*.errorDialog1_shell.width: 410

*.errorDialog1_shell.height: 200

*.errorDialog1.width: 410

*.errorDialog1.height: 200

*.errorDialog1.background: #81759d


! Resources for the Error Dialog Interface


*.helpWindow_shell.x: 500

*.helpWindow_shell.y: 280

*.helpWindow_shell.width: 875

*.helpWindow_shell.height: 500

*.helpWindow.width: 875

*.helpWindow.height: 500

*.helpWindow.background: #9d94a5


! Resources for the Control Window Interface


*.Control_Window_shell.x: 67

*.Control_Window_shell.y: 52

*.Control_Window_shell.width: 900

*.Control_Window_shell.height: 210
```

```
*.Control_Window.width: 900
*.Control_Window.height: 210
*.Control_Window.background: #9d94a5

*.form1.background: #dbc9c9
*.form1.height: 176
*.form1.width: 900
*.form1.x: 0
*.form1.y: 34

*.scrolledText1.background: #dbc9c9

*.scrolledText2.background: #dbc9c9

*.pushButtonGadget1.fontList: -Bitstream-Swiss 742-Bold-R
      -Normal--16-120-72-72-P-94-HP-Roman8

! Resources for the Title Window Interface

*.Title_Window_shell.x: 10
*.Title_Window_shell.y: 250
*.Title_Window_shell.width: 460
*.Title_Window_shell.height: 190
*.Title_Window.width: 460
*.Title_Window.height: 190
```

```
*.Title_Window.background: #9d94a5

*.scrolledList2.background: #dbc9c9
*.scrolledList2.fontList: -Bitstream-Swiss 742-Bold-R
  -Normal--16-120-72-72-P-94-HP-Roman8

! Resources for the Title Options Interface

*.titleDialog1_shell.x: 480
*.titleDialog1_shell.y: 470
*.titleDialog1_shell.width: 320
*.titleDialog1_shell.height: 190
*.titleDialog1.width: 320
*.titleDialog1.height: 190
*.titleDialog1.background: #81759d

! Resources for the Email Interface

*.email_interface_shell.x: 660
*.email_interface_shell.y: 190
*.email_interface_shell.width: 400
*.email_interface_shell.height: 240
*.email_interface.width: 400
*.email_interface.height: 240
*.email_interface.background: #81759d
```

```
! Resources for the Book Selection Interface

*.topLevelShell1.width: 430
*.topLevelShell1.height: 290
*.topLevelShell1.x: 460
*.topLevelShell1.y: 370
*.topLevelShell1.background: #dbc9c9
*.topLevelShell1.borderColor: black

*.scrolledList1.itemCount: 6
*.scrolledList1.items: Collier Condensed Encyclopedia,
       Gospels of the Bible,Knoxville News Sentinal,
       Letter P,Netlib,
       UTK's Computer Science Department's Staff
```

Table .1: XLSI Programming Information - Book Selection Interface

| *Interface* | *Widget* | *Gadget* | *Event* | *Callback* |
|---|---|---|---|---|
| Book_Selection | menu_p1_b1 | | activate | activateCB_menu1_p1_b1 |
| Book_Selection | menu_p1_b2 | | activate | activateCB_menu1_p1_b2 |
| Book_Selection | menu_p1_b3 | | activate | activateCB_menu1_p1_b3 |
| Book_Selection | menu_p2_b1 | | activate | activateCB_menu1_p1_b1 |
| Book_Selection | scrolledList1 | | singleSelection | singleSelectionCB_ScrolledList1 |

In X-Windows programming, visual objects are labeled *widgets* or *gadgets*. Specifically, *widgets* are user interface components comprised of data structures and procedures, a *gadget* is a *widget* that does not have a window of its own and must display in its parent's window. The particular procedures associated with a *widget* or *gadget* is termed a *callback*. These *callbacks* are executed as a result of a particular *event*. *Events* may be a mouse button, pressing and releasing a mouse button, pressing a certain key on the keyboard, or moving the cursor into a window. In particular, subroutines or callbacks are attached to particular widget's events.

XLSI consists of *widgets, gadgets,* and *callbacks*. Each interface (e.g. *Control Interface*), has a collection of one or more of the above mentioned window's elements. The following documents the specific *interface*, specific *widget or gadget*, and specific attached *callback*.

Table .2: XLSI Programming Information - Control Interface

| *Interface* | *Widget* | *Gadget* | *Event* | *Callback* |
|---|---|---|---|---|
| Control | menu2_p1_b1 | | activate | activateCB_menu2_p1_b1 |
| Control | menu2_p1_b2 | | activate | activateCB_menu2_p1_b2 |
| Control | menu2_p2_b5 | | activate | activateCB_menu2_p2_b5 |
| Control | menu2_p2_b6 | | activate | activateCB_menu2_p2_b6 |
| Control | menu2_p3_b1 | | activate | activateCB_menu2_p3_b1 |
| Control | menu2_p3_b3 | | activate | activateCB_menu2_p3_b3 |
| Control | menu2_p5_b1 | | activate | activateCB_menu2_p5_b1 |
| Control | menu2_p5_b2 | | activate | activateCB_menu2_p5_b2 |
| Control | menu2_p5_b3 | | activate | activateCB_menu2_p5_b3 |
| Control | menu2_p5_b4 | | activate | activateCB_menu2_p5_b4 |
| Control | menu2_p6_b1 | | activate | activateCB_menu2_p6_b1 |
| Control | menu2_p7_b1 | | activate | activateCB_menu2_p7_b1 |
| Control | menu2_p8_b1 | | activate | activateCB_menu2_p8_b1 |
| Control | menu2_p9_b1 | | activate | activateCB_menu2_p9_b1 |
| Control | | pushButtonGadget1 | activate | activateCB_pushButtonGadget1 |
| Control | | pushButtonGadget2 | activate | activateCB_pushButtonGadget2 |

Table .3: XLSI Programming Information - Title Interface

| *Interface* | *Widget* | *Gadget* | *Event* | *Callback* |
|---|---|---|---|---|
| Title | menu3_p1_b1 | | activate | activateCB_menu3_p1_b1 |
| Title | menu3_p1_b2 | | activate | activateCB_menu3_p1_b2 |
| Title | menu3_p1_b3 | | activate | activateCB_menu3_p1_b3 |
| Title | scrolledList2 | | singleSelection | singleSelectionCB_scrolledList2 |
| Title | scrolledList2 | | browseSelection | browseSelectionCB_scrolledList2 |

Table .4: XLSI Programming Information - History Interface

| *Interface* | *Widget* | *Gadget* | *Event* | *Callback* |
|---|---|---|---|---|
| History | menu4_p1_b1 | | activate | activateCB_menu4_p1_b1 |
| History | menu4_p1_b2 | | activate | activateCB_menu4_p1_b2 |
| History | menu4_p1_b3 | | activate | activateCB_menu4_p1_b3 |
| History | scrolledList3 | | singleSelection | singleSelectionCB_scrolledList3 |
| History | scrolledList3 | | browseSelection | browseSelectionCB_scrolledList3 |

Table .5: XLSI Programming Information - Document Interface

| *Interface* | *Widget* | *Gadget* | *Event* | *Callback* |
|---|---|---|---|---|
| Document | menu5_p1_b1 | | activate | activateCB_menu5_p1_b1 |
| Document | menu5_p1_b2 | | activate | activateCB_menu5_p1_b2 |
| Document | menu5_p1_b3 | | activate | activateCB_menu5_p1_b3 |

Table .6: XLSI Programming Information - Previous Search Status Interface

| *Interface* | *Widget* | *Gadget* | *Event* | *Callback* |
|---|---|---|---|---|
| Previous Status | menu6_p1_b1 | | activate | activateCB_menu6_p1_b1 |
| Previous Status | menu6_p1_b3 | | activate | activateCB_menu6_p1_b3 |
| Previous Status | menu6_p1_b4 | | activate | activateCB_menu6_p1_b4 |
| Previous Status | scrolledList4 | | singleSelection | singleSelectionCB_scrolledList3 |
| Previous Status | scrolledList4 | | browseSelection | browseSelectionCB_scrolledList3 |

Table .7: XLSI Programming Information - LSI Configuration Interface

| *Interface* | *Widget* | *Gadget* | *Event* | *Callback* |
|---|---|---|---|---|
| LSI Configuration | text2 | | activate | activateCB_text2 |
| LSI Configuration | menu7_p1_b1 | | activate | activateCB_menu7_p1_b1 |
| LSI Configuration | menu7_p1_b2 | | activate | activateCB_menu7_p1_b2 |

Table .8: XLSI Programming Information - Title Dialog Interface

| *Interface* | *Widget* | *Gadget* | *Event* | *Callback* |
|---|---|---|---|---|
| Title Dialog | titleDialog1 | | ok | okCallback_titleDialog1 |

Table .9: XLSI Programming Information - Email Interface

| *Interface* | *Widget* | *Gadget* | *Event* | *Callback* |
|---|---|---|---|---|
| Email Dialog | emailDialog1 | | ok | okCallback_email_interface |
| Email Dialog | emailDialog1 | | help | helpCB_email_interface |

# Vita

Susan Clower Allen was born in Rockwood, Tennessee on November 15, 1960. She graduated from Roane County High School in 1978 and received a Bachelor of Arts degree in Computer Science from The University of Tennessee, Knoxville in May 1985. She joined Hewlett-Packard in Atlanta in 1985. She moved back to Knoxville in 1989 where she received her Master in Computer Science from The University of Tennessee, Knoxville.