Development of A User Interface for MARIAN and CODER Systems

by

Eskinder Sahle

Report Submitted to the Faculty of Virginia Polytechnic Institute and State University in partial fulfillment of the requirements for the degree of Master of Information Systems

Approved:

Dr. Edward A. Fox, Chairman

Dr. Osman Balci

Dr. James O. Hicks

February, 1993
Blacksburg, Virginia
Development of A User Interface for MARIAN and CODER Systems

by

Eskinder Sahle
Edward A. Fox, Chairman

(Abstract)

The objective of this project is to implement a Graphical User Interface (GUI) for information storage and retrieval (IS&R) that will be easy to use, easy to maintain and easy to extend. The user interface implementation took advantage of the advanced object oriented GUI development environment provided by NeXTstep to provide an iconic interface that allows direct manipulation and supports mixed initiative dialogues.

The GUI will be used in conjunction with a family of IS&R systems at Virginia Tech: CODER (COnposite Document Expert/effective/extended and Retrieval), a test-bed that is being developed to experiment with various intelligent information retrieval techniques, and MARIAN (Multiple Access and Retrieval Information with ANnotation), an advanced on-line public access library catalog system providing network access to all facets of bibliographic information.
Acknowledgements

I would like to thank my advisor, Dr Edward A. Fox for his guidance and advice during the course of this work. I sincerely thank Dr. Osman Balci and Dr. James O. Hicks for serving on my committee.

I am indebted to Ben Cline for his unlimited support, guidance and most importantly his wit that made this work educational and enjoyable. I would also like to thank Robert France for providing extraordinary assistance during the course of this project.

Special thanks to Tim Rhodes who got me started on this project and put together the initial structure of this work. I would like to also thank Lucy Nowell for her HCI guidance. I also thank my Colleague Steve Teske for his support and assistance.

I am also deeply indebted to the following people whom in one way or another contributed to the quality of life at Virginia Tech: Babsicus , Bghad, Brd, Bimbs, Elijha, Fenge-Demame, Koni, Mimi-Bf, Nesro, Ryan, Temo-Moostache, Dr. Samush-Abush, Smallled, Turboswami, and Zerabrook,

Special thanks to my family and Nefse. And finally, thanks and praises to Egziabher, that made it all possible.
# Table of Contents

1. Introduction ................................................................. 1
   1.1 Motivation and Objectives .......................................... 1
   1.2 Advanced GUI Development. ......................................... 2
   1.3 Report Overview. ...................................................... 5

2. OPACs: An Overview .......................................................... 6
   2.1 Introduction. ............................................................ 6
   2.2 Characteristics and Classifications. .............................. 7
   2.3 Current Shortcomings and Problems. .............................. 9
   2.4 User Interface Issues ................................................. 10
      2.4.1 Types of Problems. ............................................. 11
      2.4.2 Potential Solutions. ........................................... 11
   2.5 Conclusion. ............................................................. 14

3. MARIAN System Overview ..................................................... 16
   3.1 The REVTOLC Experiment. ............................................. 16
   3.2 MARIAN System Highlights ......................................... 18
      3.2.1 The User Interface. ............................................ 18
      3.2.2 The Retrieval Server. ......................................... 18
      3.2.3 High Speed Network. .......................................... 19
   3.3 System Architecture and Design ..................................... 20
      3.3.1 System Components. ............................................ 20
         3.3.1.1 Clients ................................................... 20
         3.3.1.2 Server .................................................. 22
   3.4 Implementation .......................................................... 25
      3.4.1 Operating System and Workstation Hardware. ............... 25
      3.4.2 Communication .................................................. 27
         3.4.2.1 Interprocess Communication. ............................ 28
         3.4.2.2 Network Protocol ....................................... 29
      3.4.3 System Development ............................................ 30

4. MARIAN User Interface .......................................................... 32
   4.1 Design Philosophy and Criteria ..................................... 33
      4.1.1 Task-Specific Aid .............................................. 33
      4.1.2 Mixed-initiative Dialogue .................................... 35
      4.1.3 Modularity ...................................................... 36
   4.2 User Interface Structure .............................................. 39
      4.2.1 Processes and Interprocess Communication .................. 39
      4.2.2 Client Processes ............................................... 40
5. CODER Interface Objects for MARIAN
   5.1 Background ...................................................... 71
   5.2 Enhancements for MARIAN ....................................... 72
   5.3 Types of Enhancements ......................................... 72
   5.4 Book Display Object ........................................... 73
   5.5 Journal Article Query Creation Object ......................... 76
   5.6 Text Node Collection Display Object .......................... 78
   5.7 Development .................................................... 80

6. Lessons Learned and Future Work ............................... 82

Bibliography ........................................................... 86
<p>| Figure 1.1 | User Interface Management System Overview | 4 |
| Figure 3.1 | MARIAN Server System Components | 21 |
| Figure 3.2 | MARIAN Clients | 23 |
| Figure 3.3 | Protocol Stack | 29 |
| Figure 4.1 | User Interface Objectives and Features | 34 |
| Figure 4.2 | Interprocess Communication | 37 |
| Figure 4.3 | Client Processes | 38 |
| Figure 4.4 | Example Calling Sequence | 42 |
| Figure 4.5 | NeXT System Overview | 43 |
| Figure 4.6 | NeXTstep Components | 44 |
| Figure 4.7a | Interface Builder Palette | 49 |
| Figure 4.7b | Application Window Object | 49 |
| Figure 4.8 | MARIAN Interface Development | 51 |
| Figure 4.9 | Principal AppKit Classes and MARIAN Custom Objects | 54 |
| Figure 4.10 | Login Object | 58 |
| Figure 4.11 | Bibliographic Query Object | 59 |
| Figure 4.12 | Menu Selection to Get Query Screen | 60 |
| Figure 4.13 | Pop-up Menu | 62 |
| Figure 4.14 | Menu Selection to Submit Query | 62 |
| Figure 4.15 | Feedback Search Options | 63 |
| Figure 4.16 | Retrieval Display Object | 65 |
| Figure 4.17 | Query History Object | 66 |</p>
<table>
<thead>
<tr>
<th>Figure 4.18</th>
<th>Global Help Options</th>
<th>68</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 4.19</td>
<td>Help Options. With Specific topics</td>
<td>69</td>
</tr>
<tr>
<td>Figure 5.1</td>
<td>Bookmark Selection</td>
<td>73</td>
</tr>
<tr>
<td>Figure 5.2</td>
<td>Book display Interface Object</td>
<td>74</td>
</tr>
<tr>
<td>Figure 5.3</td>
<td>Text Creation Interface Object</td>
<td>75</td>
</tr>
<tr>
<td>Figure 5.4</td>
<td>New Note Selections</td>
<td>75</td>
</tr>
<tr>
<td>Figure 5.5</td>
<td>Figures Selection</td>
<td>76</td>
</tr>
<tr>
<td>Figure 5.6</td>
<td>Journal Article Query Creation Interface Object</td>
<td>77</td>
</tr>
<tr>
<td>Figure 5.7</td>
<td>Text Query Creation Object</td>
<td>79</td>
</tr>
<tr>
<td>Figure 5.8</td>
<td>Text Node Collection Display Object</td>
<td>79</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 4.1</td>
<td>Source Code Line Count</td>
<td>52</td>
</tr>
<tr>
<td>Table 4.2</td>
<td>Interface Specification Files</td>
<td>53</td>
</tr>
<tr>
<td>Table 5.1</td>
<td>Extended Interface Specification Files</td>
<td>81</td>
</tr>
<tr>
<td>Table 5.2</td>
<td>Extended Interface Source Code Line Count</td>
<td>81</td>
</tr>
</tbody>
</table>
Chapter One
INTRODUCTION

1.1 Motivation and Objectives

Information has become one of the most valuable commodities in today’s society. The advent of mass-computing has given the general public means to access and process voluminous information once exclusively available to a few experts and scholars. Thus, the need has emerged to design information storage and retrieval systems that will fulfill the demands of today’s information-oriented public. The challenge today, however, is to develop information storage and retrieval (IS&R) tools that are not only superior in storing and retrieving voluminous information, but that can be easily used and readily learned by the nonexpert public.

The objective of this project is to implement an advanced, modular Graphical User Interface (GUI) that is easy to use, easy to learn, and easy to maintain. The GUI will be used in conjunction with a family of IS&R systems at Virginia Tech: CODER (COmposite Document Expert/effective/extended and Retrieval), a test-bed that is being developed to experiment with various intelligent information retrieval techniques, and MARIAN (Multiple Access and Retrieval Information with ANnotation), an advanced on-line public access library catalog (OPAC) system, providing network access to all facets of bibliographic information.

Today there are various hardware platforms and development environments to implement
the front-end for sophisticated IS&R systems such as CODER and MARIAN. Previous and current efforts in implementing user interfaces for the CODER and MARIAN systems have included interfaces based on CURSES [CURS85] [CLIN91], X Windows/Motif [GHAZ91] [DAS88], and the Macintosh toolbox [WADH89]. In this project, a NeXTstep-based user interface for MARIAN and CODER is developed.

1.2 Advanced GUI Development

From the early days of computing, one of the primary objectives of research in computer science has been the improvement of human-computer interaction [THOM84]. To meet this objective, considerable effort has been invested in providing methodologies and techniques that can be used to design effective user interfaces. Nonetheless, while research in human-computer interaction is providing new and improved approaches to user interface design, there is little in the way of theory that can be applied a priori to design and implement flawless interfaces. Thus, a majority of user interface design methodologies call for experimentation and an iterative approach.

However, with many development tools available today, it is cumbersome and time-consuming to iteratively design and implement the now common graphical user interfaces. It is then not surprising that an extensive amount of effort in the application development process is devoted to the user interface. User interface developers are caught in between the increasing demand for improved interactive GUI interfaces by end-users and inadequate user interface development tools consisting of ad-hoc design methodologies, difficult to use low-level tools, and conventional programming languages.
that do not readily lend themselves to iterative development. Hence, one of the challenges in user interface research is to provide development tools that allow painless refinement as well as decrease the development time of user interfaces.

To meet this challenge, research in human-computer interaction has been directed to providing the necessary tools that support good design, rapid prototyping, and quick implementation. These types of tools that aid in the design and implementation of user interface are collectively known as User Interface Management Systems (UIMS). UIMS can be viewed as the set of software tools that sit between the user and the functional component of the application (Figure 1.1). Examples of research in UIMS can be found in [WASS85], [KASIK82], [BUXT87], [SCHU86], and [MYER87]. Moreover, an outstanding summary of UIMS concepts and current research can be found in [HART89].

The motivation behind UIMS is to free the interface developer from time consuming low-level details and concentrate on the application specific issues of the user interface [BUXT87]. A clear analogy can be drawn between Database Management Systems (DBMS) and UIMS. Just like a DBMS mediates between the programmer and the data, shielding the programmer and user from the details of searching and accessing the data, UIMS also allows the interface developer to manage and administer the user interface design, development, and implementation with minimal knowledge about the low level details. Similar to DBMS which provides consistency of technique in data access along with portability, UIMS also enforces consistency in screen layout, graphical representation and user input processing [BUXT87].
One such tool that is commercially available is the NeXTstep development environment from NeXT, Inc. NeXTstep is a highly integrated UIMS that provides an advanced graphical layout and interaction specification tool called Interface Builder, a standardized graphical display communication protocol, pre-compiled user interface elements, and a complete object-oriented development environment. As a result, NeXTstep substantially reduces the time required to iteratively develop an effective user interface. It has been one of the objectives of the author to successfully apply the NeXTstep UIMS.
Subsequently, in this project an interactive NeXTstep-based graphical user interface for
the CODER and MARIAN systems is developed.

1.3 Report Overview

This report is meant to give the reader some background information on the MARIAN
project as well as an overview of the various aspects of the project undertaken by the
author. The current chapter states the motivation and objectives of the report. The second
chapter provides a general overview of OPACs based on a literature survey by the author.
Chapter 3 introduces the architecture of the MARIAN system and highlights its important
features. The fourth chapter is an overview of the interface design and implementation. It
presents: important design issues and objectives of the MARIAN NeXTstep interface, an
overview of the NeXTstep development environment, and the implementation details of
the user interface. Chapter 5 presents a discussion of prototype user interface objects
derived from CODER to accommodate further functionalities not currently supported by
MARIAN. The final chapter summarizes lessons learned and comments on future work
required.
Chapter Two
OPACs: An Overview

2.1 Introduction

Up until the 1970's, conventional card catalogs have served as the primary device for the public to gain access to the contents of libraries [LARS91]. However, the application of powerful computer technology to Library Information Systems, originally meant for tasks such as acquisition, bookkeeping, and circulation, has provided opportunity for the development and widespread use of an automated electronic form of the traditional card catalog, the online public access catalog (OPAC) [BELK90]. OPACs allow the public to use a computer terminal to search for information about various resources like books, publications, etc. [HILD82].

The fact that OPACs grew out of Library Information Systems originally intended for acquisition, circulation and other internal bookkeeping tasks has had an effect on their design and subsequent use [REYN85]. The early OPACs were a mere extension of circulation systems designed for the exclusive use of librarians. Hence, they were difficult to use, required extensive training, and had limited capabilities. The amount of training required, although a worthwhile effort for library professionals, was not acceptable to the majority of library patrons. However, today OPACs have evolved into systems that provide improved ease of use along with extensive functionality. Current research is working on building a more robust and easier to use generation of OPACs and similar other bibliographic information retrieval systems, with advanced features that include natural language processing (NLP), highly interactive graphical user interfaces,
multimedia, advanced IS&R methods and extensive on-line help [HILD82] [HILD89] [DOSZ86].

2.2 Characteristics and Classifications

OPACs are information processing tools that belong to a family of information system tools broadly labeled as Information Storage and Retrieval (IS&R) systems. Information Storage and Retrieval systems are tools that search and retrieve records from a file of records in response to an information retrieval request [SALT83]. However, OPACs have some distinct characteristics. In particular, [BELK90] describes three distinct characteristics of OPACs:

- **They have heterogeneous infrequent users**: variation in user interest, background, computer skills, tasks, goals, age, etc.

- **They are difficult in information retrieval tasks**: complexity in articulating one's complex information need to precisely match the input requirements of a mechanical system.

- **They have a complex relationship between users' goals' and their information seeking behavior**: lack of knowledge about the relationship between the users' information seeking goals and information behavior then displayed in acquiring information at the card catalog, OPAC, or the library in general. For example, preliminary study in [BELK90] shows one frequently occurring behavior of users: going to shelves, removing books from shelves, and scanning the table of contents and/or index, and then either shelving back the book or keeping it. This suggests that OPACs can benefit users (making the table of contents and index available) by taking into account the complex
relationship between goals and behavior in the library.

OPACs can be classified into three major groups distinguished by their level of functionality, user-interface and retrieval capability [HILD82]:

1) First generation:
   • Burdensome and exacting command structure.
   • Similar information and access points as card catalogs using authors, titles and subjects.
   • Rudimentary search with exact left-to-right matching of headings.
   • Optional or automatic truncation of search strings.

2) Second generation:
   • Extended and flexible command structure.
   • Additional search methods: exact matching key-word and Boolean searching.
   • New interactive user-interface features: browsing and index displays for selecting search terms.
   • Enhanced help facility.

3) Third generation:
   • Advanced IS&R techniques: e.g., classification clustering method, probabilistic ranking [LARS91].
   • Highly interactive custom or individualized interfaces.
   • Natural language expressions.
   • Improved access through enhanced bibliographic records.
   • Context-sensitive help and error correction functions.
   • Enriched searches that integrate free text and controlled vocabulary.
2.3 Current Shortcomings and Problems

Since their appearance in the 1970s, OPACs have become a popular means of accessing bibliographic information by library patrons. A 1987 study shows that 12 percent of all academic libraries in the United States use OPACs, with another 65 percent planning to introduce them [CAMP87]. Despite the rapid popularity OPACs have enjoyed, more and more studies are showing a need for improved systems to meet users' expectations and demands [BORG86].

Literature reports summarized in [YEE91], based on various studies (which includes an extensive national effort by the Council of Library Resources) of OPACs, point out a number of deficiencies, including:

- Difficulty in expanding on query results.
- Excessive retrieval.
- Inconvenient displays: long displays that are difficult to scan, brief displays that provide extremely limited information.
- Inexperience: infrequent use that curtails the effective use of the system.
- Mechanical errors: spellings and typos.
- Spacing and hyphenated words: problems with initial and in-between spaces, inexact initials and hyphenations.
- Problems with system mechanics: understanding and remembering commands, command sequences, etc.
- Under-utilization of system features: little or no use of some powerful features that could help users better serve their retrieval needs like truncation, Boolean operations, etc.
• Awkward HELP facilities: complicated or cryptic help messages that provide novice users with insufficient assistance.

2.4 User Interface Issues

OPACs need good user interfaces to effectively serve user needs. Yet, one of the prominent reasons for user dissatisfaction is the lack of user interfaces that simplify the task of human-machine interaction for the non-expert user. To make effective use of OPACs and other similar information processing tools, the user has to invest time to acquire the skill necessary to use the system. However, studies show OPAC users avoid seeking any form of training that will improve their skills in using search systems. As a result, users tend to avoid using such systems [MISC87]. For example, studies show that a majority of OPAC users are not comfortable using command-line OPACs [MITE88] [MISC87]:

• They find them difficult to learn and remember.
• They are disinclined to use manuals and other instructional printed matter.
• They avoid using the thesaurus.
• They have problems in formulating Boolean search logic, selecting the appropriate terms, and formulating effective search strategies.
• They prefer menu driven interfaces with the appropriate help mechanism.

On the contrary, there is evidence showing that users have strong interest in utilizing user-friendly online search systems and databases that can be used with minimal training [MISC87].
2.4.1 Types of Problems

The types of problems OPAC users experience can be grouped into two major categories [BORG86-1]:

*Mechanical*: errors that are a direct result of the mechanical interaction with the system, e.g., difficulty logging into the system, typographical errors, spelling errors, etc.

*Conceptual*: problems due to lack of understanding of the search process. Examples of conceptual problems include incorrect Boolean logic formulation and poor formulation of query terms to match the retrieval system specifications (e.g., use of initials, order of surnames, first names).

Mechanical problems are important in that they determine whether the user gains access to the system, while conceptual problems determine the efficiency of the system in providing optimal service to satisfy the need of the user. In most cases, inexperience seems to be the prominent reason behind mechanical problems, where more experienced users tend to resolve mechanical errors expeditiously. In contrast, conceptual problems persist regardless of the degree of experience. Studies show frequent occurrence of similar conceptual problems across systems where the same pattern is less evident in the mechanical problems users experience [BORG86-1].

2.4.2 Potential Solutions

Various techniques for rectifying mechanical and conceptual problems have been summarized in [MITE88]:

- Providing matching aids based on morphological means (stemming, spelling checks,
matching phonetic sounds),

- Search formulation aids (extensive user help and feedback in Boolean query formulation and operation),
- Linking and terminological aids (relating retrieved items to other items in the database),
- Query expansion,
- Authority files and thesauri,
- Expert systems,
- Direct interfaces (Windows, icons, Mouse, and Pop-up or Pull-down menus).

For OPACs to be used successfully, there is a need to provide users with user-friendly interfaces that will eliminate the problems with command-line systems as well as provide ease of use and ready learnability. User-friendly interfaces provide a powerful and informative way for users to drive and control the system according to their individual needs [SHAW87]. [MEAD85] suggests three important user interface design requirements that can make user interfaces friendly (transparent) to users:

- **Cooperative**: making the task obvious with active instructional feedback.
- **Preventive**: system has to be forgiving to user errors by providing backup and recovery.
- **Conducive**: system has to be reliable, predictable and must assist users.

Systems based on such user-friendly interfaces make the users’ tasks the focus of attention rather than the intricacies of the system, and so will provide for an effective system [BORG86-2].

However, to provide effective systems there has to be a better understanding of one of the
least studied component of the human computer interaction process; the individual user. Hence, there is a need to analyze user behavior and requirements to design effective systems. Until recently, designers did not rank the individual user as an important factor in the design of OPACs [MARK84]. As outlined earlier, one of the distinguishing characteristics of OPACS is the high heterogeneity of the user base which amplifies the need to accommodate differences in information-seeking behavior.

Therefore, to be effective, the design of OPACs should account for behavioral factors that influence users [PENN86]. However, various researchers have made progress in providing new models and approaches for user interface design that take the behavioral aspect into consideration. For example, [BORG86-2] suggests a new emphasis to be placed on behavioral aspects of users.

Also, [BORG86-2] presents a set of implications for user interface design based on psychological theory and error behavior studies. [PENN86] further suggests analysis of users based on their behavior in order to uncover useful concepts that will help with design of effective user interfaces to meet their needs. [PENN86] suggests incorporating the following concepts in the design:

- Individual differences should be considered.
- The universe of users should be broken down into subgroups which are studied to identify their similarities and differences.
- Previous approaches that assumed users to be homogeneous should instead focus on individuals and sub-populations of the entire user community.
Unfortunately, most designers of user interfaces have overlooked the nature of individual behavioral needs; inclusion could contribute tremendously towards improving the user interface and thus improving the effectiveness of OPACs as an information providing tool.

However, there are other important issues aside from the user interface that need to be noted. An effective user interface will not only require a better user interface design but also needs an effective overall system design that takes the structures and relationships of the underlying data into account. This should be a strong incentive for the collaboration of systems designers and catalogers to provide better systems [YEE91].

2.5 Conclusion

Although improved user interfaces will be a positive step towards the development of a more powerful system, by itself it will not eliminate the existing shortcomings of current OPACs. Without further improvement in the underlying IS&R software to complement advances in user interface design, it will not be possible to present the public with an optimal system. [HILD87] writes:

"From the user interface to the design and structure of the database(s), today’s inverted file, Boolean query systems, recognizably enhanced by word or field proximity search methods and menu-based command interfaces, are judged to be insufficient for the future and unqualified for the label intelligent. At a deeper level, it is generally agreed that merely
attaching a "friendly" user interface on a "front-end" processor to a dumb conventional retrieval system does not make the system more intelligent or effective."

Future OPACs will have to be intelligent both in user/computer interaction and the IS&R functionalities they provide. Hence, improvements of OPACs should not be limited to palatable interfaces, but should strive to achieve improved and broader usability and smarter functionalities, both in interface and in retrieval components of the systems [HILD89].

OPACs have progressively evolved since the earliest attempts that were based on librarian bookkeeping systems. Nonetheless, it is now apparent that more work is necessary to eliminate the existing deficiencies. An essential need exists for further research to provide design guidelines and lay basic theoretical principles to improve upon current OPACs to better serve the information seeking needs of the public.
Chapter Three  
MARIAN Overview

The MARIAN project is an effort to develop a state-of-the-art online library catalog system for multiple network access to a MARC (MAchine Readable Catalog) bibliographic database. The MARIAN system employs advanced techniques in information retrieval, object-oriented programming and distributed processing, to provide users a simple, yet powerful multi-user network access to all facets of bibliographic information at Virginia Tech.

This chapter will provide a general background and technical overview of MARIAN. First an important technical forerunner to the MARIAN system is presented. Following this, the salient features of the MARIAN are discussed. In the last two sections, the overall design and implementation of top-level components is presented.

3.1 The REVTOLC Experiment

Some of the advanced IS&R principles applied in the design and implementation of MARIAN are derived from REVTOLC (Retrieval Experiment of Virginia Tech Online Library Catalog), an investigative project to test the feasibility of utilizing advanced retrieval methods on a large collection of bibliographic data. In particular, the experiment compared the applicability of Vector, Vector with probabilistic feedback, Boolean, and Extended Boolean (P-norm) retrieval methods.
To conduct the experiment, a test-bed retrieval system based on an enhanced version of the SMART [SALT75] retrieval system was developed. The SMART system was augmented by a number of other supporting technologies developed at Virginia Tech, including [FOX90]:

- A minimal perfect hashing algorithm for rapid access,
- An efficient disk-based method for building inverted files that makes efficient use of available primary memory and requires $O(n)$ time for a given collection of compressed document vectors with total length $n$,
- A form based front end that supports VT100 type terminals for the SMART system.

The REVTOLC experiment demonstrated the feasibility of applying advanced retrieval techniques to large databases. In particular, the experiment showed Vector and Vector with probabilistic feedback to be preferred over conventional techniques such as Boolean searching [FOX90].

Based on the findings of the REVTOLC experiment, the current design and implementation of MARIAN utilize Vector and Vector with probabilistic feedback as the primary retrieval methods to access a large bibliographic database. Similarly, supporting technologies developed for REVTOLC have been adopted in the development of MARIAN, including the form-based front end of REVTOLC, which inspired the design of the MARIAN user interface.
3.2 MARIAN System Highlights

The objective of the MARIAN project is to develop a high performance online catalog system that features powerful retrieval techniques. The MARIAN system is based on three important components: a user interface, the retrieval server, and high-speed networking.

3.2.1 The User Interface

MARIAN provides a simple, yet powerful user interface, running on a variety of user computers. Despite the differences in their native user interface "look and feel", the collection of interface clients will provide a homogeneous interface dialogue structure to the MARIAN server. This uniform dialogue format will allow the user to specify requests to the server as well as receive query results. Current specifications of the interface objects only support textual data.

Section 4.0 provides a detailed discussion of the MARIAN user interface.

3.2.2 The Retrieval Server

The MARIAN server provides two major services: managing the user interface clients, and fulfilling the search and retrieval requests received from the clients. The retrieval portion of the server is responsible for processing the query, accessing, retrieving and presenting search results to the client processes. The retrieval server supports
simultaneous access to all facets of the bibliographic records. Section 3.3 provides a brief description of the various server components.

To access the bibliographic databases, the MARIAN server employs advanced retrieval methods. The current implementation is based on Vector and Vector with feedback retrieval methods. The main features specified by the design include:

• Combined author, title, and subject keyword searches.
• A powerful indexing method for titles, summaries, and related text fields.
• A knowledge based approach to recognizing English words in text.
• An effective partial-match approach.
• Online assistance in locating authors and subjects.
• An effective browsing tool based on a variety of similarity measurements.

3.2.3 High Speed Network

MARIAN is a distributed system that runs on a network of computers. As part of the MARIAN system, each machine must be able to send and receive data, as well as address other machines on its own physical network connection. The Virginia Tech campus-wide ethernet network, part of the Internet network [DARP81], provides the required communication facility. The collection of network cables, routers, bridges and gateways provide physical interconnection media, data buffering, protocol translation and other related networking support necessary to provide transparent communication between components of the system.
3.3 System Architecture and Design

Although the REVTOLC experiment and related efforts [GHAZ91] demonstrated the feasibility of using advanced retrieval techniques, these pilot projects fell short of yielding a production system. One of the crucial drawbacks of the REVTLOC test-bed was that SMART, the system that served as the retrieval engine to REVTOLC, did not provide an efficient mechanism for multiuser access [GHAZ91]. In an attempt to rectify the shortfalls of REVTLOC, MARIAN incorporates a client-server architecture to efficiently support multiple users.

3.3.1 System Components

Figure 3.1 shows the major components of MARIAN. The MARIAN system architecture is one of distributed client-server computing. The server component running on a network of host computers services search and retrieval requests generated by the various clients (user interface and supporting software) runs on users’ computers. The rest of this section presents a high-level description of the system components.

3.3.1.1 Clients

Multiple user interfaces running on various types of user computers make up the client portion of MARIAN (Figure 3.2). The various clients will allow users to access the MARIAN server interactively from a variety of user systems using their standard interfaces.
Figure 3.1 MARIAN Server System Components
MARIAN will support the following types of clients:

GUI type interfaces:  Non-GUI interfaces:
- X11/Motif      • Curses (Unix & VT100 terminals via Telnet)
- NeXTstep       • VM/CMS

In addition, MARIAN will provide a batch Mail interface that will allow users to use the electronic mail facility to submit queries and receive responses.

3.3.1.2 Server

The MARIAN system software is designed as a layered distributed system. The server is made of three software layers. Each layer in turn is made up of independent software modules that perform various tasks, which include query management, database access, output formatting, user interface management, etc.

*The User Interface Management Layer*

The topmost layer is the *user interface management component of the system*. The user interface management layer consists of the formatter, query handler, parser, session manager, and UIP (User Interface Protocol) modules. The UIP module (discussed in detail in Chapter 4) manages the transactions between the user interface process (of clients) and the retrieval modules. The session manager is responsible for keeping track of user status.
It tracks the status of each session and state of each active transaction. Information kept by the session manager includes a list of active searches, progress of each search, and the number of client sessions.
The query analysis module provides two important services: query analysis and query history. The module re-organizes the submitted queries to place them in canonical form. These queries are then sent to the appropriate parser (e.g., author queries are sent to the author parser). The query handler also maintains a history of queries. Planned enhancements to the query analyzer will provide features like foreign character normalization, case conversion, and British spelling conversions.

The formatter is responsible for presenting the responses from the access methods to the client. The formatter gets the response for a user query and converts it to a format suitable to the user interface. In future enhancements, the formatter should also provide several possible choices in which the results can be displayed.

The Access Layer

The access layer consists of the combiner and other access modules that implement the various retrieval methods. The combiner module is responsible for dispatching user queries received from the user interface management layer to access modules and also combining the query result from the different retrieval modules to be passed to the user interface management layer. In the future, it may also make strategic decisions on how and where to search the various query components.

The Database Layer

The database layer is responsible for the actual access and retrieval of the bibliographic
data. The database layer is based on LEND (Large External Network object-oriented Database), an object-oriented database that incorporates techniques from database systems and information retrieval.

LEND is typical of other object-oriented databases and effectively manages complex objects, hypertext/hypermedia and semantic network operations with very large data sets [CHEN90]. The database layer with the use of LEND supports: traditional authority files, a number of inverted files for vector searches, relational databases linking the authority file to their associated bibliographic records, and document text databases containing a bibliographic/MARC record for each title that can be displayed to the user.

3.4 Implementation

3.4.1 Operating System and Workstation Hardware

MARIAN is being developed on a small network of NeXT computers with additional computers being added as needed. Initial implementation of MARIAN is on two NeXT computers based on Motorola 68040 processors. The first computer is installed with 64 megabytes of primary memory while the second workstation is installed with 32 megabytes of primary memory. Both workstations are installed with a 400 megabytes of internal and 1.2 gigabyte external hard disks.

The NeXT workstation operates using the MACH operating system. Originally developed at Carnegie Mellon University (CMU), the MACH kernel was later enhanced and ported
to run on NeXT workstations. MACH is based on a simple communication-oriented kernel that supports distributed and concurrent operation while maintaining full compatibility with UNIX 4.3 BSD [NEXT90a].

With the goal of producing a system that would meet performance goals and be extensible, two features of the MACH operating system features were deemed important: efficient message-based interprocess communication and efficient support for concurrency.

Interprocess communication between the server modules (on the same machine as well as intermachine) is accomplished through the use of MACH ports. MACH provides transparent inter-module communication. The MACH Interface Generator (MIG), a program that generates rpc [BIRR84] (remote procedure call) code for communication between processes, generates code that facilitates exchange of data between the MACH modules [NeXT90b] [TANE92]. Hence, the distributed approach will allow for straightforward extensibility of the system; that is, by simply increasing the network of workstations the various MARIAN tasks can be freely distributed on separate nodes to provide increased performance. MIG will also be available on OSF/1 (future standard for UNIX platforms), allowing for portability to future UNIX based advanced computer architectures (e.g., DEC's Alpha).

Another important feature of the MACH operating system that is essential in the development of MARIAN is the thread facility. To develop a high performance multi-user system like MARIAN requires an efficient facility to support an advanced level of concurrency. To support efficient control of processes and provide concurrency, MACH
provides an efficient unit of execution and scheduling called threads. MACH threads provide concurrent processing functionalities similar to those found in traditional UNIX processes. However, threads offer better efficiency than UNIX processes; threads can be created and scheduled efficiently, and since threads share their parents’ existing address space they provide an efficient mechanism to share data. However, the current implementation of MARIAN efficiently manages clients by making each user interface client one of multiple tasks. Within each task, the UIP spawns off threads for each client interaction; that is, each user transaction is handled by one or more threads per MARIAN module. The current implementation uses the C-threads package which allows C language access to thread processing.

Due to these features, MACH provides the necessary technology support to distribute the various software modules that make up MARIAN among several NeXT computers while efficiently integrating their functionalities.

3.4.2 Communication

MARIAN is a distributed system that relies on extensive interprocess communication over the network. The types of interprocess communication that occur can be grouped into two categories: interprocess communication among the network of modules that make up the server component of MARIAN; and intermachine communication between the server and the interface clients running on various platforms. To allow faster transfer and better control of data, all high-level interprocess communication in MARIAN is based on a client/server model utilizing rpcs. Figure 3.1 shows an overview of the interprocess communication in MARIAN.
3.4.2.1 Interprocess Communication

Interprocess communication among the network of MACH engines that make up the server portion of MARIAN is accomplished via MACH ports [TANE92] [DRAV91]. MIG remote procedure calls provide an efficient mechanism for the server modules to exchange high-level data structures [TANE92]. MACH ports along with the MIG facility provide interprocess communication facility to integrate the network of modules that make up the MARIAN server.

To facilitate intermachine communication between the MARIAN server and clients, a high-level RPC subroutine package (UIP protocol) has been implemented on top of the Internet protocols (Figure 3.3). UIP describes the standard of message formats and sequence of transaction calls used between various clients and the server. UIP is a "mixed initiative" protocol that allows both client and server to asynchronously generate requests. In addition to its own propriety protocol, UIP utilizes techniques from two other protocols: F³L (Facets/Frames/Functions Language), and Sun’s XDR (eXternal Data Representation).

In the current implementation of UIP, F³L and XDR provide the necessary data type packing functionality between the clients (user interface) and the server (user interface management component of the MARIAN server). F³L is a high-level representation language (developed at Virginia Tech as part of the CODER project) to facilitate easy exchange of information between processes [FRAN91]. XDR is a standard from Sun Microsystems, Inc. that allows a standard description of data in a machine-independent format [CORB91]. Finally, a set of protocols specifically developed for MARIAN
provides the necessary high-level data transport mechanism [CLIN91].

3.4.2.2 Network Protocol

The MARIAN system has adopted the Internet standard protocols for network communication. To provide a reliable data-stream protocol, the TCP/IP (Transport Control Protocol/Internet Protocol) protocol is used for communication between clients and the MARIAN server. Communication between processes is achieved via stream sockets.

The user interface management layer uses two other high-level Internet protocols: Telnet and SMTP (simple Mail Transfer Protocol). VT100 terminals and emulators access MARIAN through a character-based, full screen interface server. This server uses the CURSES package of UNIX to implement forms screens. User access the server through a TCP/IP terminal server or the UNIX Telnet command. Although the Telnet protocol is used, a login sequence is not required as users Telnet to a TCP/IP port directly controlled
by the server.

Similarly, the user interface management layer may provide access to mail-based clients through a mail interface using the SMTP protocol [CLIN91]. Each type of transaction is handled by a separate server: Telnet server and mail server. In turn, each of these two servers reformats requests into UIP and uses the intelligent server. The intelligent server implements the UIP protocol. The X/Motif, VM/CMS and NeXTstep interfaces can connect directly to the intelligent server.

3.4.3 System Development

Important precautions have been taken to make MARIAN a modular system that is readily extendable to meet additional requirements. Most of the system development employs object-oriented programming (OOP) techniques. The programming language used to develop the MARIAN server is C++ [STRO90], a superset of the popular programming language C [RITC91], that supports object-oriented programming.

Likewise, if the native software development environment supports object-oriented systems development, client software is also developed using object-oriented programming techniques*. For example, the NeXTstep user interface is based on an advanced, completely object-oriented, development environment using Objective-C, another C-based object-oriented programming language [PINS90]. The use of OOP allows programmers to produce a high quality modular software. In particular the inheritance feature of OOP has allowed MARIAN developers to reuse existing code while
modifying the code (object) to meet new specifications.

3.4.4 Current Status

As of this writing, the development of MARIAN is still in progress. Current implementation of MARIAN closely matches the design specification shown in Figure 3.1. On the other hand, current development efforts are progressing to provide feedback searches, the X/Motif interface, the CURSES interface, and Annotations.
Chapter Four
The MARIAN User Interface

As more and more library patrons depend on OPACs to access bibliographic information, there is a need to provide enhanced systems that meet the current demands of end users. Studies have shown that OPAC users are disinclined to use command line user interfaces and are less likely to accept the training necessary to effectively apply these systems. The same studies also indicate that most users strongly prefer robust and friendly systems that require minimal or no training over command line systems [MISC87]. Hence, it is only appropriate to take advantage of current technology to provide users with effective systems that are easy to use and easy to learn [HILD89].

In recent years the technological advances of pixel-addressable displays coupled with the widespread use of a pointing device, (e.g., the mouse), has provided new approaches to meet the demands of end users. Rapid advances in such new technologies can be harnessed to provide OPAC users with interfaces that require minimal effort-to-learn and effort-to-use. Accordingly, the effort of this project has yielded an interactive front-end to the MARIAN system based on an advanced graphical user interface environment that attempts to meet some of the user demands.

This chapter presents an overview of the MARIAN user interface development. First, the design principles and objectives of the user interface are presented. Second, the structure of the user interface is discussed. Third, the development environment used to implement the user interface is presented. Finally, an overview of the user interface is given.
4.1 Design Philosophy and Criteria

The NeXTstep interface for the MARIAN has the following general objectives (Figure 4.1):

- To provide ease of use and learnability to novice users
- To provide a powerful and efficient front-end to experienced users
- To simplify the information retrieval task

Hence, the challenge in the design and implementation of the MARIAN interface is in accommodating user diversity; that is, in providing simplicity to novice users without penalizing experienced users. Therefore, every effort has been put to make the user interface provide usability to all users and ready learnability to novice users.

To support learnability and usability, the NeXTstep user interface to MARIAN offers three important features: task-specific aid, mixed-initiative dialogue, and modularity.

4.1.1 Task-Specific Aid

One important requirement for an effective user interface is to offer a certain level of task-specific aid; that is, provide a user with the required assistance to accomplish a specific task. In particular, an effective user interface has to provide users a way to express their instructions to the system in natural terms and at the same time provide a certain level of guidance towards accomplishing users' tasks [SHNE92]. To provide task-specific aid, the MARIAN user interface should provide users with two types of task-specific assistance: a simple and easy to use direct manipulation user interface and an
online help facility. These design features of MARIAN's NeXTstep interface can reduce the user's effort-to-learn and effort-to-use the MARIAN system.

Through the use of direct manipulation, the MARIAN interface allows users to interact with the system using recognizable objects borrowed from their day-to-day experiences.
like buttons, windows, sliders, menus, etc. For novice users, graphical user interfaces based on real world metaphors provide a natural analogy with their experience, providing ready learnability. Moreover, direct manipulation interfaces eliminate memorization and extensive training by relieving users from the effort required to remember cryptic commands. Direct manipulation also provides experts with the potential to rapidly execute a wide range of tasks [SHNE92].

Similarly, a well-designed online help facility provides learnability to novice users that otherwise would have been difficult [SHNE92]. To provide learnability and usability, the current implementation of the MARIAN help facility provides general help to assist users in learning more about the system’s features (e.g., an overview of MARIAN’s general features and capabilities) and explicit help on specific topics that assist users on specific features (e.g., how to submit a query). Furthermore, future plans call for an extension of the help facility to include context sensitive help that will provide users with point-of-need help based on the immediate context of the user’s action. Similarly, the help system would greatly benefit from some form of online tutorials that can reduce users’ effort-to-learn and effort-to-use the system [SHNE92].

4.1.2 Mixed-Initiative Dialogue

Human-computer interaction employs various forms of dialogue. According to [HUTC86], the types of human-computer interaction dialogues can be grouped into two major modes: **sequential mode** and **asynchronous mode**. In the sequential approach, the user typically participates in the dialogue by providing sequential instructions to the system through the use of a command language (e.g., data entry, traversal through
menus). On the other hand, user actions like window re-sizing, clicking on a button, or dragging a graphic object across the screen are examples of asynchronous dialogue. The MARIAN interface features a form of asynchronous dialogue that allows multi-threaded interaction where a user can initiate multiple tasks independently. Multi-threaded dialogue loosely imitates human-to-human conversation where an individual can initiate multiple threads of action and conversation.

Similarly, a user of the MARIAN interface can initiate multiple threads of dialogue with the system. For example, a user can compose a query and submit a search task and then switch to browsing a previously retrieved collection without waiting for the search task to complete. At the same time, the system could be posting warning messages to the user while processing the query instruction from the user. In this mode of dialogue, known as *mixed-initiative*, not only is there a bi-directional flow of information, but each side can take the conversational lead [FRAN92]. By providing users the power to pursue multiple threads of conversation, the system provides enhanced usability by freeing users from the inflexibility of traditional sequential interaction.

### 4.1.3 Modularity

The NeXTstep user interface to MARIAN was developed in adherence to strict modularity guidelines enforced by object-oriented programming. Great care has been taken to provide *dialogue independence*; that is, to separate the user interface component from the application (server) component of the system. With dialogue independence, design and implementation issues that affect the user interface component are isolated.
from affecting the application component and vice-versa [HART89]

**Figure 4.2 Interprocess Communication**
Hence, dialogue independence allows multiple interface development for the same host machine that can be tailored to a different groups of users [HART87]. Moreover, it allows the development of different types of host specific interfaces and simplifies portability in a distributed computing environment.
With current user interface design methodologies calling for iterative refinement, the advantage of developing modular interfaces is obvious. The ability to iteratively refine the user interface to deliver an effective user interface has been affordable primarily as a result of the modular design and implementation of the MARIAN user interface.

4.2 User Interface Structure

The MARIAN user interface is a separate process that runs in parallel with the MARIAN server. The interface process is designed to be an intermediary between the application process and the user, presenting user actions to the server and server action to the user [FRAN92]. Communication between the user and the server is provided through a collection of interaction objects managed by the user interface. The interaction objects are self contained units. These objects belong to a class with an inheritance hierarchy and encapsulate the required data structure and functions (methods) necessary to access data. Examples of interaction objects used include text editing windows, a variety of menus, dialogue boxes, and alert panels.

4.2.1 Processes and Interprocess Communication

Figure 4.2 shows the structure of the user interface and interprocess communication. On the client workstation two MARIAN processes service the user on behalf of the server: the User Interface Manager and the Client-UIP process. On the server, the UIP Handler process and Session Manager process are primarily responsible for managing the
interaction with the client process (user interface).

4.2.2 Client Processes

One of the primary processes on the client side is the User Interface Manager (Figure 4.3). The User Interface Manager provides the environment necessary for the user to communicate with the server via the interaction objects, and the server with the user. To accomplish these tasks, the User Interface Manager for NeXT computers relies on the services provided by the NeXTstep Window Server. The Window Server process is responsible for processing mouse and keyboard events as well as the rendering of all images on the screen. After accepting keyboard and mouse events, the Window Server dispatches the keyboard and mouse events to the Interface Manager process. Once the Interface Manager process receives the events (user commands) from the Window Server, it uses the Client-UIP process to convert the function requested by the user into a remote procedure call to execute on the server.

Figure 4.4 shows an example of the calling sequence used to search the database. The Window Server reads the command from the keyboard or mouse and passes it to the Interface Manager. Depending on the type and content of the interaction object used, the Interface Manager will map the command to one of the callbacks to the server. The callback will be initiated in the form of a remote procedure call that is executed on the server.

Similarly, the User Interface Manager can receive instructions from the server via the Client-UIP process (Figure 4.2). In this case, the calling sequence described above is
reversed, i.e., the Client UIP receives a message from the server which is mapped into a call to a function within the User Interface Manager. Depending on the type of instruction received, the Interface Manager will in turn instruct the Window Server process to render one of the interaction objects, that will communicate the server's message to the user.

4.2.3 Server Processes

As discussed in Chapter Three, the server is made up of several processes. Among the various server processes, the UIP handler and the Session Manager processes have direct responsibility to manage and process all interactions with the client processes. The UIP handler is responsible for receiving all messages via a TCP/IP socket stream from the UIP client process. The message received (rpc call) is decoded as a function and a list of arguments by the UIP Handler. A function that corresponds to the decoded name and arguments is then executed in the UIP handler. The function executed in the UIP Handler adds a session number to the list of arguments for session identification and invokes a MIG remote procedure call to the Session Manager. As depicted in Figure 4.4, the userChoiceO message received from the UIP client process will execute a similar function userChoiceO in the server UIP handler. The userChoiceO function in the UIP handler will then invoke the userChoiceO function in the session manger via a MIG rpc.

Similarly, when any of the underlying server processes tries to communicate with the client, the particular process interested in servicing the client will invoke a function in the
VIP handler via a MIG call. The UIP handler in turn will execute a remote procedure call to the client UIP process via a TCP/IP socket stream.

4.2.4 Interprocess Communication

To accomplish the above sequence of events the various processes involved use one of two interprocess communication methods: MIG messaging and TCP/IP based virtual
circuits (stream sockets) [COME90]. The interprocess communication between the client processes and the server processes is accomplished through stream sockets in the *Internet* domain. On the other hand, communication among the various NeXT client processes, i.e., Window Server, Interface Manager, and UIP client is accomplished via MIG messages (discussed in Chapter 3).

4.3 The NeXTstep Development Environment

NeXTstep is a development and a graphical user interface environment for NeXT workstations [WEBS89]. Figure 4.5 gives a system overview of the NeXT operating environment.

![Figure 4.5 NeXT System Overview](image)

To the general user, NeXTstep provides a consistent easy-to-use graphical user interface.

In particular, the NeXTstep user interface provides [NEXT90a]:
- Consistency across all applications
- Natural user interface that uses real world metaphors
- User control, i.e., users are free to choose which application and window they will work in and customize the interface environment according to their needs

To developers, NeXTstep provides a highly integrated, object-oriented graphical user interface environment. As illustrated in Figure 4.6, NeXTstep is made up of four software components: Window Server, Interface Builder, Application Kit, and the Work Space Manager [NEXT90a]. The Work Space Manager is the graphical user interface to the NeXT’s operating system. In other words, "The Workspace Manager is UNIX with a new face" [CLAPP90]. It provides users with essential support to perform various tasks
including file management, launching applications, or working with disks: mounting, initializing, naming, etc. The remaining three components provide the essential support required to develop graphical user interfaces: display management, event processing, layout capability, and development toolkit support.

### 4.3.1 Display Management and Event Processing

In the NeXTstep environment, the Window Server is responsible for the creation and display of user interface objects and processing of user generated events. The Window Server creates and owns windows - the fundamental type of interface objects under NeXTstep. Windows are a rectangular region where every application's graphical and textual output is displayed; i.e., every running application requires at least one window associated with it. To users, windows are a focus of input to type text in or manipulate user interface objects (e.g., buttons, sliders, etc.) using a mouse. Since most NeXTstep applications are interactive in nature, they need to display information on the screen. Therefore, most applications require the services of the Window Server and are said to be clients of the Window Server.

To display information on the screen an application sends the Window Server drawing instructions. These drawing instructions are sent in the form of PostScript code [PERR88]. The PostScript code is interpreted into drawing instructions which the Window Server executes in the appropriate window owned by the application that sent the instructions.

A majority of NeXTstep applications are event driven; that is, application reaction
depends on user events. In addition to managing all of the display processing on the screen, the Window Server also provides event processing support required by all running applications. Each user keystroke and mouse based action is captured and sent to the appropriate application by the Window Server. The application receives user’s action results from the Window Server in the form of an event. Examples of events received by an application include cursor positioning, indication of which window the action took place in, etc.

4.3.2 Graphical Layout and Interaction Specification Tool

One of the most important productivity tools for user interface developers is a graphical layout tool which reduces the user interface prototyping process to a simple step of graphically assembling interface objects through direct manipulation. NeXTstep provides the interface developer a graphical interface layout tool called Interface Builder. The interface builder enables the interface developer to interactively design and specify the graphical layout and functionality of a graphical user interface. In particular, Interface Builder provides the developer with the ability to rapidly prototype the interface and apply almost effortless iterative refinement of the interface.

Using Interface Builder, the developer can create the necessary interaction objects. To graphically design the interface layout, Interface Builder provides the developer with a palette of user interface objects that includes buttons, menus, sliders, etc. (Figure 4.7a). To design an interface, the developer simply drags a user interface object and places it anywhere on a window that represents the application’s user interface (Figure 4.7b). Interface objects have built-in capabilities to process user events such as key strokes and
mouse actions. In addition, the objects handle their own basic display functions. For example, the text object displays the characters corresponding to a user's key strokes, a slider repositions and redraws itself when dragged, a button knows to highlight itself when clicked on, etc. The specification of the assembled interface objects is then saved to an interface file with a .nib extension.

In addition to providing the ability to graphically assemble the interface layout, the Interface Builder also provides the ability to create connections between the graphically assembled interface objects. Since message exchange is the primary form of integrating the various self contained objects like buttons, sliders, menus, etc., Interface Builder provides a graphical way to form functional interconnection between objects. Once these connections are specified as a result of a simple graphical action, the objects can send messages to one another. For example, the developer can connect a slider object to a text field object. Such a connection will establish the functional relationship between these two objects where the text field is continuously updated when the slider is dragged.

4.3.3 ToolKit

NeXTStep provides a powerful development toolkit called the Application Kit. The Application Kit provides the developer with a collection of object-oriented, ready-made building blocks that can be easily integrated to create very complex user interfaces. All of the objects from the palettes window (Figure 4.7a) are part of the Application Kit object classes. Using Interface Builder the various interface objects from the palettes window can be graphically assembled to construct the desired interface. Aside from the interface objects available through the Interface Builder pallets, the Application Kit also provides
additional objects that are used in interprocess communication, printing, and media integration.

All of the interface building blocks provided by the Application Kit are implemented as classes of objects using the Objective-C [COX91] development language. One of the powerful features of objects from the Application Kit is the ease of extensibility they provide through inheritance. Objects from the Application Kit Objects are fully extensible using Objective-C. For example, the developer can create a new class that inherits from the Button class. This new class will have all the functionality provided by the Application Kit’s Button class. The developer can further extend the functionality of this base class by adding additional capabilities thereby saving time and effort that would be necessary in traditional approaches that do not readily support reusability.
Figure 4.7a Interface Builder Palette

Figure 4.7b Application Window Object
4.4 Implementation of the User Interface

The implementation process of the user interface was typical of software development efforts under NeXTstep. The implementation process adhered to a disciplined object-oriented approach consisting of several iterations of experimentation and refinement. The development was accomplished using the three important tools provided by the NeXTstep 2.0 development environment: Interface Builder, the Application Kit and Objective-C. The graphical layout capability provided by Interface Builder coupled with the highly modular objects developed using Objective-C have permitted successive refinements of the user interface without painstaking changes that would have been inevitable in traditional approaches. Figure 4.8 illustrates a typical application development process under NeXTstep that consists of three general phases: graphically assembling the user interface, developing custom objects, and debugging and testing the user interface.

4.4.1 Assembling the User Interface

The user interface development process began by defining the various windows and functionalities required. This initial step was accomplished using Interface Builder where the various user interface objects were graphically assembled to create the required windows and panels. The visible part of the interface created by Interface Builder (each interface window and panel) was then saved as a special file with .nib extension. All interface files created using Interface Builder have a .nib extension. The MARIAN user interface consists of ten .nib files (Table 4.1).
Applications with several screens benefit from storing different parts of their interfaces in separate interface files. Using multiple interface files the application can improve performance by not creating all the interface objects indiscriminately at start up. Instead, the interface objects are created upon request. Although a slight performance degradation might be encountered when trying to access an auxiliary interface object that has not
been created, the delays are negligible compared to imposing a performance degradation to all users upon invocation. Moreover, storing the interface in separate files provides a simple yet powerful approach to create a given part of the interface (or multiple versions of it) an indeterminate number of times.

Table 4.1 Interface Specification Files

| BFQCWindow.nib:     | Query Formulation window |
| MARIAN.nib:        | Information Display window |
| RDWindow.nib:      | Main menu |
| TDWindow.nib:      | Retrieval Set Display window |
| TCDWindow.nib:     | Text Display window |
| MLWindow.nib:      | Choice List Display window |
| TCDWindow.nib:     | Informational Question window |
| TCWindow.nib:      | Login window |
| YNQWindow.nib:     | Text Creation window |
| answer soliciting window |

4.4.2 Developing Custom Objects

After creating the visible components of the interface, the next step involves development of the interface objects that are not visible to the user. These objects manage the visible part of the user interface (windows, panels, menus, etc.) created using Interface Builder. In particular, this step involves the implementation of all custom objects that are specific to the application under development.

These include objects that:

- mediate between the server and the user interface,
- track the state information,
• load specific interface files upon request, and so forth.

### Table 4.2 Source Code Line Count

<table>
<thead>
<tr>
<th>Specification Files</th>
<th>466</th>
<th>65</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFQCWindow.m</td>
<td>105</td>
<td>27</td>
</tr>
<tr>
<td>CLDCustomCell.m</td>
<td>426</td>
<td>47</td>
</tr>
<tr>
<td>CLDWindow.m</td>
<td>143</td>
<td>27</td>
</tr>
<tr>
<td>CustomCell.m</td>
<td>34</td>
<td>5</td>
</tr>
<tr>
<td>ErrorObj.m</td>
<td>105</td>
<td>18</td>
</tr>
<tr>
<td>IDWindow.m</td>
<td>426</td>
<td>19</td>
</tr>
<tr>
<td>IQWindow.m</td>
<td>143</td>
<td>5</td>
</tr>
<tr>
<td>IDWindow.h</td>
<td>34</td>
<td>5</td>
</tr>
<tr>
<td>MARIANController.m</td>
<td>92</td>
<td>18</td>
</tr>
<tr>
<td>MARIANController.h</td>
<td>1550</td>
<td>19</td>
</tr>
<tr>
<td>MARIAN_main.m</td>
<td>14</td>
<td>5</td>
</tr>
<tr>
<td>MLWindow.m</td>
<td>90</td>
<td>21</td>
</tr>
<tr>
<td>RSDWindow.m</td>
<td>580</td>
<td>61</td>
</tr>
<tr>
<td>TCWindow.m</td>
<td>366</td>
<td>41</td>
</tr>
<tr>
<td>TCWindow.h</td>
<td>85</td>
<td>41</td>
</tr>
<tr>
<td>TDWindow.m</td>
<td>344</td>
<td>8</td>
</tr>
<tr>
<td>TDWindow.h</td>
<td>85</td>
<td>8</td>
</tr>
<tr>
<td>YNQWindow.m</td>
<td>85</td>
<td>8</td>
</tr>
<tr>
<td>YNQWindow.h</td>
<td>4358</td>
<td>580</td>
</tr>
<tr>
<td>total</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Interface Files</th>
<th>65</th>
</tr>
</thead>
<tbody>
<tr>
<td>BFQCWindow.h</td>
<td></td>
</tr>
<tr>
<td>CLDCustomCell.h</td>
<td>27</td>
</tr>
<tr>
<td>CLDWindow.h</td>
<td>47</td>
</tr>
<tr>
<td>CustomCell.h</td>
<td>27</td>
</tr>
<tr>
<td>ErrorObj.h</td>
<td>5</td>
</tr>
<tr>
<td>IDWindow.h</td>
<td>18</td>
</tr>
<tr>
<td>IQWindow.h</td>
<td>19</td>
</tr>
<tr>
<td>MARIANController.h</td>
<td>200</td>
</tr>
<tr>
<td>MLWindow.h</td>
<td>21</td>
</tr>
<tr>
<td>RSDWindow.h</td>
<td>61</td>
</tr>
<tr>
<td>TCWindow.h</td>
<td>41</td>
</tr>
<tr>
<td>TDWindow.h</td>
<td>41</td>
</tr>
<tr>
<td>YNQWindow.h</td>
<td>8</td>
</tr>
<tr>
<td>total</td>
<td>580</td>
</tr>
</tbody>
</table>

The custom objects were developed using Objective-C. Since the custom classes are primarily concerned in managing the visible part of the interface, they need know how to allocate memory. By making all of the custom objects subclasses of the Object class (Figure 4.9) they inherit the ability to allocate memory from the Object class.
Figure 4.9 Principal AppKit Classes and MARIAN Custom Objects
The user interface consists of various independent classes and interface files. Each class requires two types of files: interface files (with .h extension) and implementation files (with .m extensions) listed in Table 4.2. The interface files contain the description of the object where variables and methods are defined. Likewise, the implementation files contain the implementation details of all methods.

The primary function of most custom objects is to manage the collection of user interface objects specified by the interface files listed in Table 4.1 (.nib files discussed in Section 4.2.2). These custom classes are in turn managed by the MARIANController class. On behalf of the user interface the MARIANController class

- initializes the user interface,
- coordinates the custom classes,
- receives messages from the server, and
- sends messages to server.

4.4.3 Source Files and Modules

As mentioned earlier the user interface has two types of files: the interface files and the class files. Table 4.1 lists the interface files used by the user interface. Similarly Table 4.2 lists all of the class files associated with the user interface.

The size of the executable is 1.12 Mb. The user interface consists of a little over 6000 lines of source code.
4.4.4 Debugging and Usability Testing

Once the custom objects were implemented, all the user interface objects including the standard objects from the Application Kit were compiled and linked together. Interface Builder provides a project management environment that creates and maintains a makefile containing information on how to build an application from the various source files. This file is used by the standard Unix make program to create the application's executable file.

The user interface has gone through three distinct phases of testing. Since most of the NeXTstep user interface components were developed ahead of the MARIAN server, the basic functionality of the user interface was initially tested using simulated callback and callin routines in lieu of the server. After testing the interface on a stand alone system, the software required to support interprocess communication between the NeXTstep client and the server across the network was subsequently integrated with the user interface manager and tested. The simulated callback and callin functions were once again used over a TCP/IP socket stream for further tests.

The third stage of testing includes usability testing. This stage of the testing process is providing extremely useful feedback to help with the evolutionary refinement of the system. With the current server version supporting most of the basic retrieval task (based on a small subset of the MARC bibliographic records) as well as management of interface clients, usability testing of the system is currently underway. The usability testing will provide potential users of the system with the opportunity to evaluate the user interface. The evaluation provided by the users has provided valuable suggestions and
guidance to the development team to improve upon the current design of the interface.

Based on subsequent usability testing an attempt will be made to iteratively refine the design of the user interface. From the initial usability testing the following important shortcomings and additional requirements were identified [NOWE92]:

1. Provide feedback on MARIAN to notify the user of the following activities:
   - notification after invoking MARIAN that the system has been successfully invoked.
   - notification when there has been a system failure.
   - notification of search status after query submission.

2. Reorganize menu options
   - rename some of the menu options to better express the actions they accomplish
   - restructure some of the menu hierarchy to reflect task organization

3. Provide better information on how the system operates
   - communicate the ordering principle of the retrieval set

4. Provide more control to users
   - maintain user settings and selections (e.g., window sizes, results format/order)

5. Simplify the manner users accomplish tasks
   - eliminate case sensitivity
   - provide easily accessible buttons for frequent and rudimentary tasks
   - provide user selectable ordering of results (e.g., rank order vs. alphabetical order)

Since the implementation of the user interface is independent of the application component, a majority of changes in the user interface will not affect the server component of the system. Moreover, since the various interface windows and panels are
implemented as separate objects and files, modifications can further be localized to a particular object or file. Furthermore, using Interface Builder, most of the layout changes are easily accommodated.

4.5 The MARIAN User Interface Screens

The MARIAN user interface is made up of several different screens that provide the necessary functionality required to communicate with the underlying retrieval system. When first invoked the MARIAN user interface displays an optional Login Object (Figure 4.10).

![Login Object](image)

Figure 4.10 Login Object

Upon successful login, the user can use MARIAN via the main menu which is always displayed on the upper left hand side of the screen. After completing the login procedure,
the user can proceed to search the bibliographic database. In this section an overview of three of the most important screens, for query formulation, retrieval display, and help system, will be presented.

4.5.1 Query Formulation

An important function of any retrieval system is to provide a facility to formulate queries. MARIAN provides the Bibliographic Query Window, a query formulation screen that reflects the organization of bibliographic information in an online catalog (Figure 4.11).

As illustrated in Figure 4.11, the query formulation window is based on a form structure that allows users to enter their query by filling in a form. This form based approach to query formulation is expected to be more intuitive and therefore easier to use [SHAH88]. To bring up the Bibliographic Query Window, the user needs to select the New Query option from the main menu (Figure 4.12).

The Bibliographic Query Window (Figure 4.11) consists of five scrollable text objects (fields) that represent the different types of concepts that can be used to formulate a query.

• Author
• Title
• Subject
• Notes
• Combinations of Title, Subject, and Notes
The first text object is used only for author specification. In this text object, the user is given the option to provide a full or partial description of the author(s). The next three text objects provide the user with several alternatives in defining queries.
The current implementation requires users to formulate queries by furnishing information in one or more of the following categories:

- Title
- Subject
- Notes
- Combination of the above: Title and Notes, Title and Subject,
- Any part of description, i.e., Title or Subject or Notes

The aim of these three text objects is to allow the user to specify word(s) to be searched in the Title, Subject, Notes, or a combination of either or all fields of the bibliographic records. The selection of one of the options (Title, Subject, Notes, or a combination of either or all) is set through the pop-up menus provided next to each of the three middle text objects (Figure 4.13). When the user selects one of the selections from the pop-up menu, it will be displayed, and sets the way the query is defined for all words entered in that particular text object.
MARIAN also provides the user with an option to formulate queries based on the date of a given work using the last text object field shown in Figure 4.11. Once the user formulates a query by filling out the form provided by the Bibliographic Query Window, a search can be initiated in one of two ways; the user can click on the search button (Figure 4.14) on the Bibliographic Query Window or use the Perform Search option (Figure 4.11) from the main menu.

For example if the user is searching for a book on the subject of "information retrieval system", s/he can enter the words "information retrieval system" in one of the three middle text objects and set the corresponding pop-up menu to Subject. The user then initiates the search either by pressing the Search button on the Bibliographic Query Window, or by selecting Perform Search from the main menu. The query is presented to the server which retrieves the required information, formats it and presents it back to the
user interface client. The user interface uses a display object to display the retrieved information on the screen [Figure 4.16]. The retrieval display is discussed in detail next.

4.5.2 Retrieval Display

The titles found in response to a query are displayed in the Retrieval Display Window shown in Figure 4.16. The retrieval display window has two scrollable display objects: the top half, which we will refer to as Works Found Display Object, contains the titles of the documents retrieved as a result of a query. The bottom half, which we will refer to as Selected Items Display Object, provides further detail on the selected title(s) from the top half of the window.

As illustrated in Figure 4.16, the Works Found Display Object has a list of titles that have "information retrieval system" in their titles. Further detail on one or more of the retrieved works can be obtained by simply selecting the item(s) of interest. The list of
titles in the top half are mouse selectable items. A single click on a title or holding the left mouse button down and dragging the cursor across more than one of the titles selects the title(s). Upon selection of title(s), the corresponding detail on the selected title(s) is displayed in the Selected Items Display Object of the Retrieval Display Window.

In addition to displaying retrieval results, the Retrieval Display Window can also be used to initiate a feedback search based on one or more titles. To initiate a feedback search for a single title, the user can select any one of the retrieved titles by clicking on a particular title and then selecting the item labeled From current document menu from the sub-menu of Choose (Figure 4.15).

Likewise, the user can make multiple selections for feedback search using the check marks provided on the top-half of the Retrieval Display Window. The check marks corresponding to each title are used to mark titles that are of interest for feedback search. Clicking on the check marks selects the corresponding title for feedback search and places an "x" within the check mark box giving the user a visual reminder of the titles marked for feedback search.

Once the user marks the titles, feedback search based on multiple titles is initiated by selecting the From selected documents menu selection under Choose item on the main menu. Unfortunately, the current version of the server does not support feedback search. However, development is in progress to support feedback search in the next version of the server software.

The Retrieval Display Window does not always display all of the works found by a
particular query. Rather, only a partial list, 20 or less of the presumably best works found, are returned by the server for display.

Figure 4.16 Retrieval Display Object

The maximum number of documents returned by the server at any given display session is determined by a user selectable option setting from the main menu. However, if the user would like to see more works that were retrieved in response to the query but not displayed, the Find more using this query selection (Figure 4.15) can be used to display more document titles.

An indicator above the object that displays the titles provides the user with the number of presumably best works that are retrieved and can currently be browsed using the display.
Since the mixed initiative style of dialogue (discussed in Section 4.1.2) allows users with any number of queries, it is possible to submit new ones before results from previous queries are presented. To help the user manage a number of queries, a history of queries is maintained by the Query History Object (Figure 4.17). By double clicking on one of the query titles listed in the Query history, the user can display a previously prepared query to edit or resubmit.

4.5.3 The Main Menu and MARIAN Commands

The NeXTstep user interface guidelines [NEXT92] dictate that all applications have a main menu. This main menu bears the name of the application and holds the commands for the application. The MARIAN interface employs this approach with a few exceptions where commands have been implemented both as part of the main menu and as part of the specific window they are applicable to. A notable exception had to be made in the
case of the **Search** command, the command that submits a query to the server and initiates the search process. The reason behind violating the NeXTstep user interface guidelines was conclusive evidence derived from usability testing that showed users overwhelmingly prefer the **Search** command to be accessible from the query formulation window rather than solely from the main menu.

Otherwise, most of the "look and feel" of the MARIAN NeXTstep user interface adheres to the NeXTstep user interface guidelines. When the MARIAN application is launched, the user interface main menu always appears in the upper left hand corner of the screen. The NeXTstep guideline provides various reasons for this approach including [NEXT90a]:

- It is less likely to block the view of other windows which are usually centrally located.
- Since the menu always appears at a well known position on the screen, users will know where to find it.

Moreover, the main menu provides all of the possible commands that can be used in relation to the MARIAN system. Some of the commands or menu options in the main menu are standard to most NeXTstep applications. These commands currently available in the NeXTstep MARIAN client include:

**Info**: Displays a panel containing basic information about MARIAN.

**Edit**: Provides commands to edit current selection in a editable document. The standard commands are Cut, Copy, and Paste.

**Preferences**: Displays a panel that allows the user to customize MARIAN. Currently the
address and port to the MARIAN server can be set using the preferences option. In addition the current implementation also allows the user to select the maximum number of documents to be displayed in response to a query.

**Hide**: Hides all the windows and the main menu associated with the MARIAN application. To retrieve the windows, it provides a button on which the user can double-click.

**Window**: Provides commands affecting windows that are associated with MARIAN. The commands under the Window menu allow the user to miniaturize a window, close a window, and tile all windows associated with MARIAN in an orderly manner.

**Quit**: Terminate the MARIAN application.

### 4.5.4 Help Facility

MARIAN provides an online help facility. The **Help** option in the main menu is what the user selects to get help from the system. It provides a sub-menu which contains a list of options in the form of menu items. See Figure 4.18.

This list of menu items is made of important meaningful categories that provide assistance necessary to accomplish tasks under MARIAN. For example, if the user is requesting help while using the Bibliographic Query Form, the help options available will include specific help on the Bibliographic Query Window as well as help on more general categories (Figure 4.19). Help actions from the user interface are not processed locally by the User Interface Management process (Section 4.2.2). Rather, a help request in the interface results in a callback to the server specifying the interaction object where the help has been initiated from.
Based on the interaction object specified in the callback, the server formats the appropriate help message and returns it to the user interface. The user interface uses a non-modal display object to display the help message.

Online help that offers concise descriptions could be ineffective for infrequent novice
users, [SHNE92]. To overcome this shortcoming, the MARIAN help facility provides a
descriptive message of a paragraph or more for the various subjects covered by the help
system.
Chapter Five
CODER Interface Objects for MARIAN

5.1 Background

With the explosive growth of electronic information today, there exists a need for tools that can organize and provide easy access to the voluminous information available. To address this need, CODER, a multi-year research project at Virginia Tech has been investigating advanced methods in IS&R that can provide heterogeneous users with an effective tool to access large and diverse types of information.

In particular, the CODER project investigated the utility of artificial intelligence techniques to improve information storage and retrieval methods. Initial efforts focused on the analysis and representation of heterogeneous documents which are commonly found in the context of computer based message systems such as electronic mail digests. Subsequent efforts under project INCARD [FKCF91] adapted the CODER system to provide integrated access to a large collection of bibliographic citations, a large thesaurus of medical terms, and a full text document of cardiology. An analysis and complete description of CODER and related efforts are beyond the scope of this chapter. For an in depth discussion on the design, architecture and details of development of CODER the reader is referred to [FOX86], [FOX87a], [FOX87b], [FC87], [FF86], and [FKCF91].
5.2 Enhancements for MARIAN

With an increase in the volume of information that is becoming available on electronic media such as CD-ROMs, WORMS, etc., MARIAN users could benefit immensely from full text retrieval capability that would allow them to access library resources such as books, periodicals, citations, journal articles, and similar other free text information. Therefore, drawing on previous experience gained from CODER in integrating bibliographic, full-text, and thesaurus data, the MARIAN system can be adapted to provide free text analysis, indexing, storage, and retrieval. To support free text storage and retrieval, the user interface development effort under this project has extended the base MARIAN interface objects to include interface objects derived from CODER. However, as of this writing, the MARIAN server does not support full text storage and retrieval. Hence, the interface objects, at best, can only serve as functional prototypes.

5.3 Types of Enhancements

The CODER/INCAR interface objects provide additional functionalities beyond what is available through the native MARIAN interface objects described in Chapter Four. One important function that will be available through the additional interface objects adapted from CODER/INCAR is the ability to support hypertext. Furthermore, the interface objects will provide two additional functions not available through the native MARIAN interface objects (Chapter Four): a full text display object and a journal article query. Support for full text access along with hypertext capability is expected to provide users with an effective information exploration tool. The rest of this chapter will briefly describe these additional interface objects derived from CODER.
5.4 Book Display Object

The Book Display window is a prototype for a hyperbook display (Figure 5.2). It allows the user to browse a book according to the normal flow of text. The display allows the user to set the types of font and style markings. Beyond normal text browsing, the display also provides specialized markers for bookmarks and footnotes. For example, paragraphs of interest can be marked for later referrals by clicking on the Place Bookmark item from the Bookmark main menu selection (Figure 5.1). Clicking on the place Bookmark item selection brings up a modal window that allows users to enter the bookmark string of their choice. The string provided as a bookmark is then inserted at the current cursor position. The list of bookmarks can be enabled/disabled by selecting the Show/Hide Bookmark item under the menu.

Figure 5.1 Bookmark selections
Chapter 2
The NeXT User Interface

Changes made for the current release of NeXTstep affect the information presented in this chapter. For details see:
/NextLibrary/Documentation/NextDev/Notes/UIUpdate/UIUpdate.rtf

This chapter discusses the NeXT user interface from the programmer's point of view. It's meant to serve as a bridge between your experience as an end user of the NeXT computer and your experience writing applications for other end users.
• It explains the user interface and introduces some of its rationale so that it will be easier for you to understand.

Figure 5.2 Book Display Interface Object

The display also allows users to take notes while browsing the textbook. Clicking the New Note selection under the Notes main menu item displays a Text Creation Object (Figure 5.3) to enter notes. The note specified in the Text Creation Object is then inserted...
at the current cursor position. As part of future plans, this screen can be further modified for use with bibliographic data to handle annotation.

The user can also look at figures and illustrations using the Book Display Object. To examine the available figures, the user needs to click on the **Figures** main menu selection (Figure 5.5) which brings up a list of figures and tables available under the current topic. The user can display the figure of interest by clicking on a title from the list. The **Previous** and **Next** buttons allow sequential browsing page-by-page of the topics in the current textbook.

![Figure 5.3 Text Creation Interface Object](image1)

![Figure 5.4 New Note selections](image2)
5.5 Journal Article Query Creation Object

The Journal Query Creation Object is similar to the Query formulation object discussed in Chapter 4. The Journal Query Creation Object, however, is structured to accommodate query formulation specific to a collection of journal articles. As illustrated in Figure 5.6, the query formulation window is based on a form structure that allows users to enter their query by filling a form. To bring up the Journal Query Creation Object, the user needs to select New Journal Query option from the main menu.

As shown in Figure 5.6, the Journal Query Creation Object consists of four scrollable text objects (fields) that represent the different concepts that can be used to formulate a query.
• Author
• Subject
• Journal title
• Year

Figure 5.6 Journal Article Query Creation Interface Object
The first text object is used only for author specification. In this text object, the user is given the option to provide a full or partial description of the author(s). The next text object provides the user with two alternatives in defining the search scope of the query: in the title only, in the abstract only, or in both title and abstract.

Selection of one of the options (Title, Abstract, or a combination of all) is set through the pop-up menus provided next to each of the three middle text objects (Figure 5.6). When the user selects from the pop-up menu, the selected option will be displayed and the way the query is defined for all words entered in that particular text object is set.

The Journal Query Creation Object also provides the user with an option to formulate queries based on the date of a given work. The last text object field shown in Figure 5.6 allows the user to formulate a query based on the date of the work. As shown in Figure 5.6, the Journal Query Creation Object is similar to the Bibliographic Query Formulation Object discussed in Chapter Four. Although the Bibliographic Query Formulation Object can be used to query information, the user can be presented with a query object that reflects the underlying structure of information (in this case journal articles) being queried.

Once the user formulates a query by filling out the form provided by Journal Query Creation Object, a search can be initiated by one of two ways: the user can click on the search button (Figure 5.6) on the Bibliographic Query Window or use the Perform Search option from the main menu.
5.6 Text Node Collection Display Object

Each Text Node Collection Display Object represents a hypertext node. A node is a fundamental unit of information in a hypertext document which ideally covers one concept [EJ91]. To display a node via the Text Node Collection Display Object, the user first enters a query term using the Text Query Creation Object shown in Figure 5.7.

In response to the query submitted, the Text Node Collection Display Object shown in Figure 5.8. is displayed. The Text Node Collection Display Object is made up of two scrollable display objects: The text display object found in the top half of the window contains the textual description of the node retrieved as a result of a query. The bottom half provides a list of short description strings.

Each entry in the list of strings serves two distinct purposes: each entry in the list is a link anchor, i.e., a visual indicator of a cross reference to another node, and each string serves as a descriptor to the node cross referenced by the link. When a user clicks on any one of the links in the list, the node cross referenced by the selected link is displayed. Once the original node is replaced by a new node, the user can re-display the original node by clicking on the Previous button. Similarly, clicking the Next button displays the next sibling of the last one seen.
Definition: any of a class of animals of the genus lupus, including the grey wolf and the European timber wolf.

Figure 5.8 Text Node Collection Display Object

5.7 Development

The additional interface objects described above have been developed. However, since the server does not support any of the functions these objects provide, the objects have not been integrated and tested. The objects were developed based on the specifications presented in [FRAN92]. For the Journal Article Query Creation Interface Object and Text Node Collection Display Object future integration and testing only requires compiling and linking the version of the user interface that supports these objects. However, the Book display object requires further code development to support all of its functionalities discussed above. In particular, routines that implement bookmark insertion,
selection and display and note insertion need to be developed.

Table 5.1 and 5.2 provide a summary of source files of the additional interface objects.

<table>
<thead>
<tr>
<th>Table 5.1 Extended Interface Specification Files</th>
</tr>
</thead>
<tbody>
<tr>
<td>BookDisplay.nib: Book display window</td>
</tr>
<tr>
<td>JAFQWindow.nib: Journal Query window</td>
</tr>
<tr>
<td>TNSDWindow.nib: Hypertext Node display window</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Table 5.2 Extended Interface Source Code Line Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Specification Files</td>
</tr>
<tr>
<td>BookDisplay.m</td>
</tr>
<tr>
<td>JAFQWindow.m</td>
</tr>
<tr>
<td>TNSDWindow.m</td>
</tr>
<tr>
<td>total</td>
</tr>
<tr>
<td>Interface Files</td>
</tr>
<tr>
<td>BookDisplay.h</td>
</tr>
<tr>
<td>JAFQWindow.h</td>
</tr>
<tr>
<td>TNSDWindow.h</td>
</tr>
<tr>
<td>total</td>
</tr>
</tbody>
</table>
Chapter Six
Lessons Learned and Future Work

The objective of this project was to develop an interactive graphical user interface for a family of information systems: MARIAN and CODER. During the process of realizing the goals set forth at the onset of the project various valuable lessons have been acquired. In particular, the project has provided a chance to work with state-of-the-art tools that include an advanced graphical user interface development environment (NeXTstep), an advanced operating system (Mach), and a powerful object oriented programming language (Objective-C).

Moreover, it has provided the opportunity to experience and learn about the intricacy of harnessing advanced computing technology in an attempt to create a system that serves the users effectively. In developing an interactive graphical user interface an important point realized is the challenge and subsequent effort required to accommodate heterogeneous users with varying level of skills, backgrounds, interests, etc. The challenge of trying to accommodate various suggestions and approaches to meet the objectives set forth at the onset of the project has directly translated to several iterations and experimentation. Long and satisfying hours have been spent in refining the interface in order to provide richer functionality, ease of use, and ready learnability. Indeed through successive refinements a number of desired functionalities have been accommodated.

It should be carefully noted, however, that the level of iterative refinement and
experimentation was pragmatic only as a result of abiding by disciplined object-oriented development both in the design and implementation of the user interface. Hence, the lessons and experiences gained in understanding and applying object-oriented technology have proven invaluable. Previous efforts similar to this project have produced good results in implementing similar user interfaces for earlier versions of CODER and MARIAN [GHAZI91] [WADH89] [DAS88]. However, one of the most important virtues of this project that somewhat distinguishes it from previous similar efforts is the strict object-oriented paradigm followed throughout the development stages.

Although the user interface has undergone several iterations and testing, there will be a need for further usability tests based on the production version to be deployed for use by the public. These tests will invariably reveal additional shortcomings. Hence, further experimentation and refinement is necessary to eliminate existing deficiencies and software bugs.

Although future usability tests will identify overlooked needs and deficiencies, there are some important enhancements that are currently identified and deemed important but have not been incorporated to the current version of the user interface. Among the essential future extensions required are:

- enhanced context-sensitive help,
- universal "undo" command,
- user controlled ordering of retrieval sets, and
- online manual and brief tutorial.
Enhanced Context-sensitive help

Context-sensitive help will provide users with detailed help depending on where they are in the system. For example, context-sensitive help in the text field of the query form window will provide instructions on how to complete that particular field, and how to configure the popup menu to the desired type of search. Most importantly, the instruction can be augmented with specific useful examples. To novice users this form of assistance could prove invaluable.

Universal "undo" command

As one of the aims of the user interface is to provide an easy to use front-end to complex information systems, it is important to provide features that allow users to recover from mistakes they are bound to make. Preventive features are especially important among novice users that are apt to make repeated errors. Hence, a feature that allows some form of undo can be of significant help to novice and infrequent users. In particular it will be beneficial to implement a universal undo feature to the MARIAN interface that will recover a query that has been formulated and was changed or lost due to user or system error.

Online manual and brief tutorial

For a system to be effective it needs to provide active instructions that make the task at hand obvious. Online manuals that provide rich instructions can be effective in achieving this goal. Online manuals that briefly present an overview of the system along with brief tutorials and examples of the commonly performed tasks can greatly increase the overall effectiveness of the user interface.
User controlled ordering of retrieval sets

Finally, the current implementation orders the retrieval set based on relevance to the original query. However, providing the user control over the order of the retrieval set can simplify the process of pin-pointing titles relevant to the user. Possible orderings that can enhance the effective use of the system include: alphabetical ordering (author and title), reverse chronological ordering and call number ordering.

Although the sort functionality could be implemented as part of the user interface, it will be a better design decision to delegate this responsibility to the server. By making the server responsible for the necessary retrieval set ordering, redundancy of sort routines in each and every interface client can be avoided.
Bibliography


[MEAD85] Mead, J.A. "Friendly or Frivolous?" *Datamation*, 3(1):96-100, April 1985


89


