Database Manager for Envision

by

Kaushal R. Dalal

Project Report submitted to the faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

in

Computer Science

APPROVED:

Prof. Edward A. Fox, Chairman

Prof. Lenwood S. Heath

Prof. Dennis G. Kafura

August, 1994

Blacksburg, Virginia

Database Manager for Envision

by

Kaushal R. Dalal

Committee Chairman: Prof. Edward A. Fox

Department of Computer Science

(ABSTRACT)

As multimedia systems and digital libraries increase in popularity, it becomes increasingly important to develop systems able to store and retrieve multiple formats of data, provide an intuitive, usable interface and allow concurrent execution of multiple sessions. This project concentrates on the issues of database management – data loading and retrieval of data – for the Envision System. A version of the database manager supporting approximately 100,000 records is implemented.

ACKNOWLEDGEMENTS

I wish to express gratitude to my advisor, Dr. Edward Fox for his perseverance and guidance throughout my work. I have learnt a lot from him apart from the technicality of the education. I thank Dr. Kafura for being on my committee and providing useful suggestions.

I sincerely thank Dr. Heath who supervised my work and gave me excellent guidance on design and development. I thank the Project Envision team, and especially Scott, Bill, Robert, Eric and Guillermo; it was great fun working with them. I also thank Steve Teske and Ben Cline in the Computing Center for useful suggestions throughout my work. I thank the NSF for providing financial support and allowing us to carry out the research on Project Envision in the grant entitled, "A User-Centered Database from the Computer Science Literature" (IRI-9116991).

I am indebted to my parents and family members (Darshana, Inuphoi and Mukund-mama) for their constant encouragement, affection and inspiration throughout my educational career. Without their backing and moral support, I would never have made it this far. I dedicate my project to them.

Ramana, Rajiv and Tejas have been good friends. I have always found support from them during harsh moments.

These acknowledgements would be incomplete without mentioning a person who has made my life at Tech a colorful, enjoyable and an unforgettable experience – Aarti Khanna. A great friend and an excellent person, she has been a tremendous source of inspiration, enthusiasm and support to me.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Digital Libraries

Interest in a digital library of the future has grown in part because of three basic facts. The first is the human need for information, especially information that is timely and relevant. The second is the accumulation of information from research groups, publishers, associations, professional organizations, and academic institutions. Third, with the advances in technology in networking, human-computer interaction, information access, hypermedia, and multimedia, it has become possible to provide easy, efficient and fast access to gigabytes of information.

A digital library utilizes these recent technologies to reduce barriers to the creation, dissemination, manipulation, storage and reuse of information by individuals and groups. It not only provides its users with the traditional capability of searching for a book, or journal or publication of interest, but also gives a browsing capability to walk through the information space to find things that are of interest. A digital library attempts to break barriers of information that are available in different media forms. Hence, information in audio, graphic or video forms should be stored and represented as easily as information in text forms. A digital library simplifies activities of librarians through automatic indexing. Conventional cataloging systems assign only a few keywords for each catalog entry; a digital library supports long user queries, natural language search capabilities, ranked retrieval, and user ability to assign weights to query terms [GAA94].

Project Envision, a research effort at Virginia Tech, aims at building a prototype digital library for computer scientists. Envision, "a user centered database from the computer

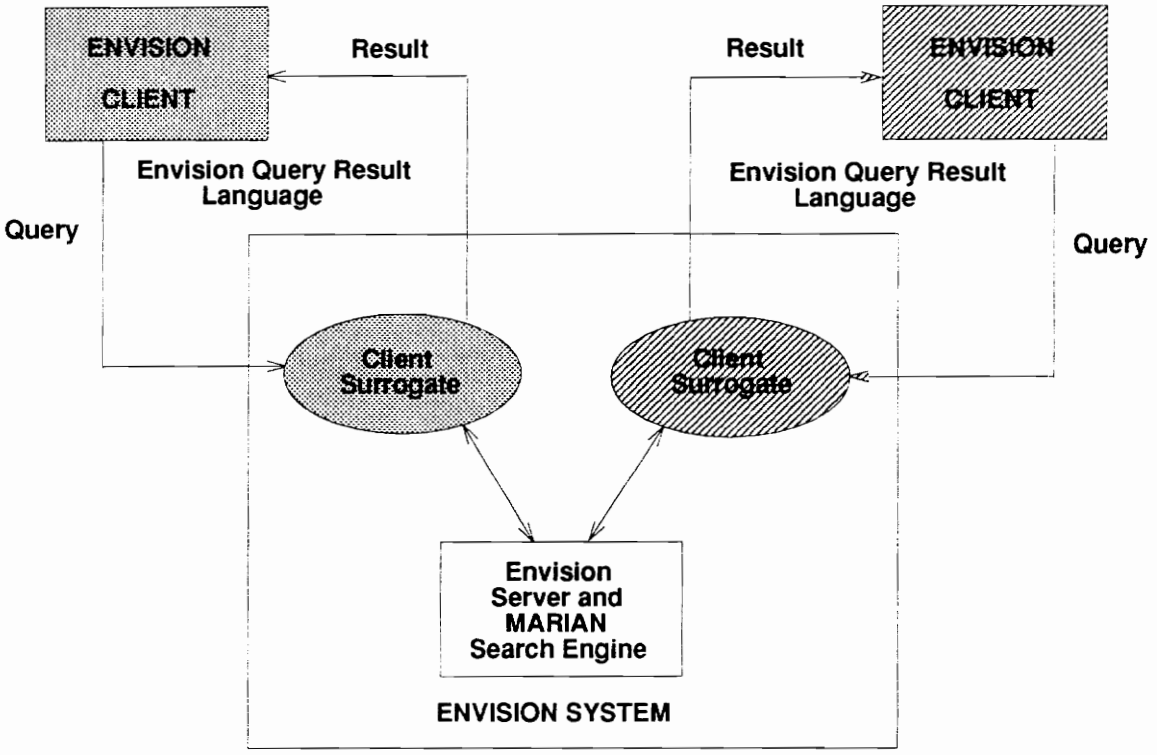

Figure 1.1: Top Level Architecture of Envision System

science literature", takes its ideas from user-centered design, multimedia databases and information retrieval [BCF93] [FHH91] [Fox92].

1.2 Project Envision

A goal of Project Envision is to solve some of the important research problems relating to digital libraries, especially those relating to information storage and retrieval, human-computer interaction, and electronic publishing [GAA94] [BCF93]. Accordingly, it follows a set of principles which should be the basis for future digital libraries – representing document components as objects, providing links among objects, organizing the link hierarchy, user-centered development and advanced ranked retrieval techniques [FFK91] [Rao93] [FHN93].

The Envision system is a multithreaded client-server system where multiple clients can search or browse the database consisting of a large collection of text, audio, video and graphic information. Figure 1.1 shows the top level architecture of the Envision system. A simple communication protocol is used for communication between a client and the Envision server [Hea94].

The Envision client has been developed considering many needs of the user community. Potential users – librarians, computer scientists and students – were interviewed and the user interface was iteratively refined to achieve innovative query formulation and search result screens. The query window can be applied in multiple ways to search for an item of interest to the user. It also displays a list of authors in the database, so that users can browse and select from it. The result window presents the result set in the form of bubbles. The result window includes several features: the documents are shown to users in 2-D plot form with the X-Y axes labeled by variables such as author name, relevance or year; users can mark documents to be relevant or irrelevant; users can save the query and corresponding result set and reuse it at some later time. [FHN93].

Figure 1.2 shows the developer's view of the Envision system. The shaded part in the figure represents the document database functions and is the focus of this report. The server consists of an Envision intermediary to manage control flow in the server, a search system to search for a query and a document database to store the documents. The Envision search system builds upon earlier work on Multiple Access Retrieval of Information with ANnotations (MARIAN), developed by the Virginia Tech Computing Center and Department of Computer Science [FFS93], CODER, an advanced information retrieval system based on artificial intelligence and expert system techniques [FFK91] and the Large External Network Database (LEND) system [Che92]. MARIAN forms the search engine for Envision. It is a distributed concurrent search system consisting of a variety of searchers communicating with each other. The document database stores the documents in Standard Generalized Markup Language (SGML) format. The client surrogates communicate with the client to obtain the query and return the result set. The Envision intermediary

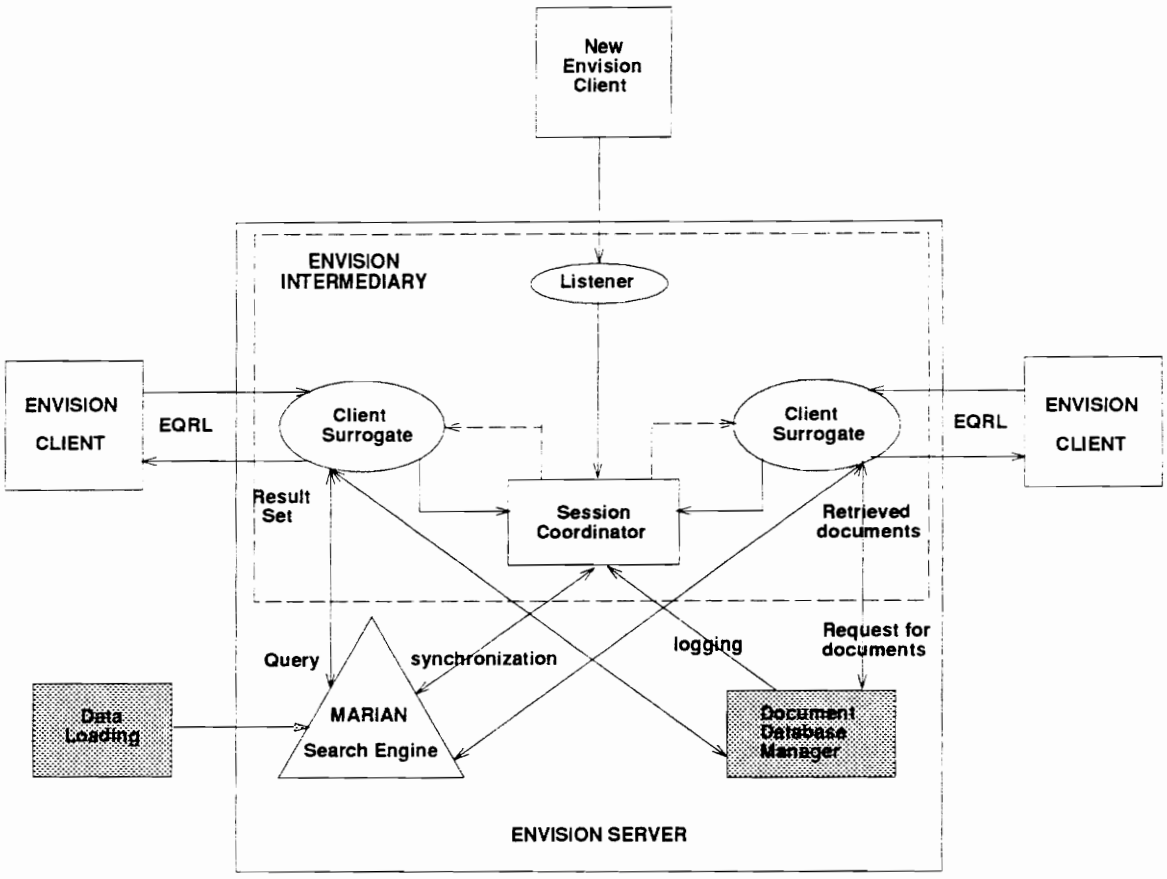

Figure 1.2: Developer View of Envision System

(within the server) manages the client surrogates. The current version of the server runs on a NeXTstep system; however plans are underway to port the system to a DEC Alpha in the future.

A communication protocol, that works with the Envision Query Result Language (EQRL), is used by both the client and the server. Those modules interpret the queries and result sets [Hea94]. The communication protocol is synchronous in nature in that, after issuing a query, the client waits for a result set from the server. However, users can abandon the search at any time.

Envision supports access to a large collection of computing literature by a rich representation of the contents of that literature. In the near future, Envision will have hypertext

links connecting interesting document features to related points of interest in the database. Documents are stored in a manner intended to maximize and simplify reuse of their contents. SGML provides the standard framework for the descriptive markup required to achieve such goals. The content elements of articles may be thought of as nodes in a large information graph, connected by structural, referential, and semantic links. The HyperText Markup Language (HTML), a subset of SGML, provides a standard for document presentation [Gol90] [Her90] [LCo93].

Once a user identifies one or more relevant documents, the user may view the document with Mosaic. Users can thus use Mosaic to view the full-text of the document, or present a video, or look at a graphic image. Mosaic serves the dual functionality of being a document viewer for Envision and a hypermedia tool over the Internet [LCo93] [Lee93]. Plans are underway to provide connectivity to the Envision database. The objects such as authors, institutes and conferences would be linked and hyperlinks supported by Mosaic then would be used to find more interesting things stored in the database.

1.3 Scope of the Project

This project focuses on the development of the database module in the Envision server. Its functions are briefly identified as follows:

The database manager performs the functions of analysis, storage and retrieval of SGML records. Each document in Envision has a corresponding entry for it in SGML. In response to a query, the server presents the user with a profile of the retrieved documents. If the user wants more information about an item, the user is presented with a full version of the document. In particular, the database manager performs the following functions:

- analysis of SGML records for indexing purposes,
- loading the database into our version of MARIAN,
- storing SGML records with their length and address information, and

- retrieval of SGML records depending on their bibliographic identification number and packaging them in EQRL to be sent over the network to the client.

1.4 Organization of Project Report

This project report is organized as follows. Chapter 1 gives a brief introduction to Project Envision and digital libraries. Chapter 2 outlines the development of Project Envision. It describes the development of the client, developing the DTDs for articles and bibliographies, and the Envision data. It also describes the related work of porting the Envision system to OSF Alpha, SGML to HTML translation and MARIAN searchers. Chapter 3 describes the database manager in detail. Finally, Chapter 4 concludes the project with some suggestions for future work.

Chapter 2

PROJECT ENVISION

This chapter is divided into two sections. Section 2.1 reviews the input documents for this project. It describes the various data formats incorporated in Envision and the document type definitions for articles and bibliographies. Section 2.2 discusses the related work in the development of the client and the server. In particular, it focuses on the Envision client interface, Envision search engine and the issues of providing connectivity to the database.

2.1 Documents

This section describes the background work, particularly of the development of Document Type Definitions. It also describes the input Envision documents and the process of translating them to SGML.

2.1.1 Envision Data

It is essential to establish a standard for data storage. Envision stores all text data structured in SGML. The source for Envision data may be a *publisher*, a *department*, an *institute* or a *professional organization*. A document obtained from such a source can typically be classified as either a *journal article*, an *article in a book*, an *article in periodical*, an *article in proceedings*, a *book*, a *booklet*, a *chart*, an *editorial*, a *letter*, a *manual*, a *Master's thesis*, a *Ph.D. thesis*, a *review*, a *technical report*, an *artwork form* or a *general entry*. The source of the article typically classifies the document into one of the above classes. Also, a document can be either *bibliographic information*, a *bibliographic entry with abstract*, a

full-text article or a multi-media document such as *video*. An automated technique is needed to translate and store these documents in SGML.

For the initial version of Envision, we have Association of Computing Machinery (ACM) data in text format. In this project, approximately 100,000 of these documents were loaded into the Envision server. The documents provided by ACM can be categorized in two major classes: bibliographic entries (with or without abstracts) and articles. Articles are full-text articles containing front matter, an abstract and body. These bibliographic entries and articles represent information about articles, books, journals, conferences, proceedings and technical reports in human-computer interaction, computer graphics, compilers, information retrieval, operating systems and so on. A detailed list of document sets in the Envision database, their source, document size, and other statistics can be found in Appendix A.

The ACM documents in their raw form are not necessarily in SGML. They are rather in formats such as *refer*, *bibtex*, or *standard markup language (SML)*. An automated procedure is thus needed to convert each raw format to SGML [Gol90]. For the full-text documents in text format, a translator based on the article Document Type Definition (DTD) is used. Similarly, for the bibliographies, a bibliographic DTD is used for translation [ASU86] [Br93A] [Br93B]. However, the Envision database is not limited to SGML data. Data in various other formats such as *TIFF, JPEG, MPEG, PostScript, or Portable Document Format* will be added to the Envision database. Since PostScript is a page-description language defining font sizes and word spacing for words within a document, it is practically impossible to translate a document in PostScript to a corresponding SGML form. Similarly, it is infeasible to translate a video document to SGML format. Thus, for all these types of documents, when a document is added to the database, a bibliographic (or full-text) entry (in SGML) corresponding to that document is also stored in the database with a pointer to the document in its original format. This description is used for indexing purposes and to present users with a *profile* of the document in the result set. However, if they find a document interesting, they can query for the whole document and the Envision system presents the original document using the most suitable document presentation tool.

Thus, for example, Envision might show a *PostScript* document in *ghostview*, run a *MPEG player* for *video*, run *Acrobat Exchange* for *PDF* documents, and run *xv* for *TIFF* images.

2.1.2 Document Type Definitions

The Envision Project has developed a DTD that supports all types of Envision text documents, currently articles and bibliographies. A DTD defines the context-free grammar for a document in the form of rules. A rule is defined using the notation:

lhs — *rhs*

where lhs (left hand side) is the name of the element being defined and rhs (right hand side) is the content model. The — symbol suggests a rule in a context-free grammar [Br93A] [ASU86].

These rules define the document structure. They encapsulate the information specific to that document type such as: precedence levels, occurrence of tags within other tags, and type of the document. Some examples of valid document types are a *journal article*, an *article in a book* and a *book*. The DTD allows *paragraphs* to occur within *sections*, *subsections* or *abstract* only. A typical structure of a full-text article is as shown in Figure 2.1 [Br93A]. Thus, an article may consist of a bibliography (Mybib), the front matter (FM), body and the back matter (BM). Further, front matter can consist of the title of the article, author name(s), date, and abstract. An abstract can contain one or more paragraphs. Similarly, the body of the article may have sections. Finally, the back matter consists of acknowledgements, zero or more bibliographies and appendices.

Formatters are available which display the document based on the style sheet for the document format. Parsers are written which validate the document for the type of DTD and extract information stored between tags in a document. A detailed description of our DTD can be found in [Br93A] and [Br93B]. For Envision, the DTD defines grammars that encompass the variety of formats in the source documents.

The source text documents (as described in the previous section) in different formats such as *bibtex*, and *refer* have to confirm with the Envision DTD; they are then translated

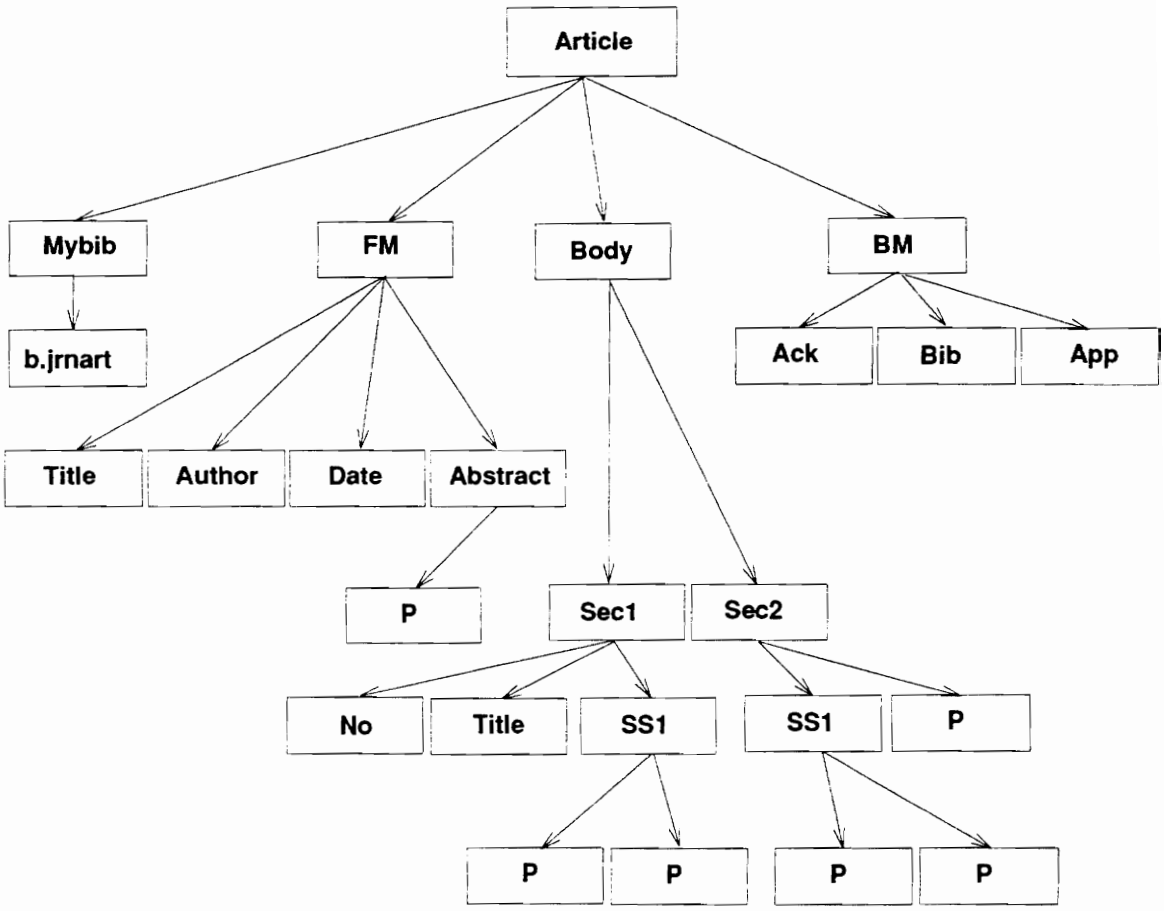

Figure 2.1: Structural Definition of an Article

to SGML with tags inserted at the proper place [LMB92]. We use the DTD to extract information from the Envision source data. The data loading process is explained in detail in Chapter 3.

2.2 System Development

This section describes the work in this project as it relates to the development of the Envision system. It, in particular, describes development of other parts of the system such as the Envision client, SGML-HTML translation and development of MARIAN as a search engine, as well as porting MARIAN to a Digital Equipment Corporation OSF Alpha.

2.2.1 Developing the Client Software

It is extremely important to have a system that is not only functional, but can also effectively communicate with the user. Hence, design and development of the Envision client has followed a user-centered paradigm [Shn92] [HHa93] [BCo91]. Professionals in the areas of computer science and information retrieval were interviewed to identify user expectations. Following is a short listing of some of the features users want to see:

- access to the information retrieval system from the workstation in their office,
- ease of structuring queries,
- meaningful feedback for unsuccessful searches,
- an interface easy to use so that a novice can quickly become a power user [HHa93] [Shn92],
- access to multiple forms of information (abstract, brief description, full text) for each document,
- ability to explore the literature for the information on objects (author, book, institute, organization) of interest.

- powerful filters to let users easily locate highly relevant items, and a
- browsing facility.

Responding to interviewees' concern that an information retrieval system must be accessible from their offices, design of the client was based on the premise that the Envision user interface will run as a process on a user's desktop computer, communicating with the Envision retrieval system via a network. The interface design provides flexible use of varying configurations of monitors, both in size and resolution. The Envision interface specification calls for separate windows or groups of windows for each of the major phases or types of interaction with the Envision system [FHN93]. These include [FHN93]:

- Query Window (with four special query fields and a query history),
- Search Results Windows (Graphic View and Item Summary), and
- Browsers.

Query Window

The Envision Query Window gives users the benefits of natural language query formulation. Users can either form a new query or access previously completed (old) queries and the results of the related searches provided [FHN93].

The Query Window offers a user three ways to create new queries:

- by entering document descriptors in four new query fields for authors, title words, words related to the content, and words found in other parts of the document,
- by editing earlier queries, and
- by combining results of previously completed searches, using set operations.

Search Results Window

The Envision design attempts to use innovative graphical presentations of search results to users. It uses two windows: the Graphic View Window and the Item Summary Window to show search results.

1. **Graphic View Window:** The central concept in the design of the Graphic View Window is the idea of visualizing a document as a bubble or a group of documents as a cluster in the Envision database. Results of search are thus presented as a scatter plot of icons [Rao93]. Users can customize their configuration along the X and Y axes. By manipulating icons, users can perform a variety of functions such as [FHN93]:

   - view basic bibliographic information in the Item Summary Window,
   - examine the abstract or full-text description of the document,
   - use the document represented as the basis for a feedback search,
   - print or save selected information pertaining to the item.

2. **Item Summary Window:** Textual summary information is displayed in the Item Summary Window, when an icon in the Graphic View Window is selected [FHN93].

2.2.2 MARIAN Searchers

Envision uses MARIAN as a search engine for efficient retrieval of data. MARIAN is an object-oriented multi-threaded information retrieval system developed by the Virginia Tech Computing Center and Department of Computer Science. This section gives a brief overview of the MARIAN objects, MARIAN system architecture and the user interaction protocol (UIP).

MARIAN Objects

MARIAN adopts the techniques of morphology-based matching, query expansion, authority files, linking and terminological aids. MARIAN identifies different classes of objects

such as a *string component*, *authority*, *document*, *link* and an *inverted file*. Each of them is briefly explained below:

1. String component: Strings include words found in the *Collins English Dictionary*; there are also variations such as derivations of a word or irregular forms derived from the source. These are useful for morphological processing carried out during document and query indexing.

2. Authority: The authority objects are obtained from keywords, terms and personal or corporate names. The authority objects can be related to one or more *subject* categories. These objects are normalized and linked to corresponding documents, so that matches can be identified.

3. Document: These are documents themselves, which are preserved in their entirety.

4. Link: MARIAN models the document database as an information graph with nodes representing documents, author names or subjects and links between them to define the relationship. The link databases store information about documents and their links.

5. Inverted file: For efficient retrieval, an inverted file is produced for each subject, name and text field in the SGML record. For example, in Envision an inverted file is produced for authors, titles, subjects and bodies of the document. An inverted file stores each term in the database along with its *term weight* in the database, the number of occurrences of the term, and a list of documents in which the term occurs. Each term in the database is categorized into one of three subclasses: ONE for terms that occur in a single document, FEW for terms that appear as a standard list of postings and MANY for terms that commonly appear in documents. In the case of a term that occurs just ONCE in the database, the document number corresponding to the term is stored. If a term occurs FEW or MANY times then a pointer to

the posting list or the posting set of documents in which the term occurs is stored, respectively.

System Architecture

To provide increased parallelism, MARIAN adopts the concepts of distributed processing and concurrency [Cli93]. Parallelism is achieved by concurrent execution of threads which can be described as low-cost, efficient processes. The MACH operating system provides efficient interprocess communication among different MARIAN servers through Mach Interface Generator (MIG) calls. Communication between clients and the MARIAN server follows the User Interaction Protocol (UIP) developed at Virginia Tech. UIP consists of two layers: The upper layer corresponds to user interface objects [Sah93]. The lower layer is a symmetrical, remote procedure call protocol for transport of user interface objects between the MARIAN server and its clients. UIP runs as an application layer protocol above TCP/IP. The implementation of UIP is thread-safe to support concurrency in the MARIAN server [BKL93] [TRG87].

A brief description of the different MARIAN servers involved in searching follows:

1. UIP Handler: It is the interface between the user interface manager and the rest of the system. It accepts UIP messages originated from user actions and translates them to MIG calls on the server module. Conversely, it also maps calls for user interaction beginning within the server to user interface objects for the correct client [Sah93] [Cli93] [BKL93].

2. Session Manager: It oversees activities within the system. It is responsible for graceful termination of internal processing for a stopped session. It maintains session status. Other MARIAN modules report their progress to the session manager [Fra92].

3. Parsers: Various parsers such as the *text parser* and an *author parser* translate the user's representations of objects to system representations. A user's representation is

usually a sequence of characters. The system representation is rather a term vector or the number of occurrences of a term or a combination of both [Fra92].

4. Searchers: Different searchers send the *Combiner* a retrieved set, and the result set size. The number of items returned is bounded above by the number of items requested by the user. However, searchers continue the search until the probability of a new document retrieved in response to the query changing the result set is less than 5 %. As new documents are retrieved, updates are made to the result set (which consists of up to the k best items, where k is the number of documents requested by the user). Finally the documents not in the result set are ignored [FFS93].

5. Combiner: The combiner controls the *Searchers*, scheduling their performance of different types of searches [Fra92].

2.2.3 Objects and Links in Envision

Envision aims to provide a hyperspace for users to find things of interest. Users might, for example, want to find more information about an author, an institute, or they might want to look at an article that is quoted by the current article. It is therefore important to identify classes of objects users would be interested in and establish links among objects [D^+91] [BOS91]. The Envision database can thus be visualized as an information graph with nodes with unidirectional or bidirectional links among them [AHe94].

Since the documents are represented using SGML, it would be ideal to have an SGML viewer, which would display a formatted document based on the document type. Envision, however, uses Mosaic to present documents with the HyperText Markup Language (HTML). HTML is based on SGML but supports fewer tags than SGML and is basically used as a presentation markup language. Documents are translated on the fly to HTML when a user asks to view the document.

2.2.4 Porting to Alpha

I made an attempt to port MARIAN to an Alpha. However, there are several problems which need to be addressed before a port to Alpha can be successful.

1. The Alpha doesn't seem to support *nameserver* lookup upon which MARIAN heavily relies to find its servers.

2. Some kind of mapping mechanism is needed to map the C thread library available on NeXT to the P threads library on an Digital Equipment Alpha.

3. The availability of GDBM on Alpha or a compatible database has to be explored.

4. A compatible GNU C++ compiler running on Alpha with its libraries is needed.

2.3 Summary

This chapter described the Envision data sources, and background work for translating and storing documents using SGML, subsequent to developing DTDs. It also described the development of client software for Envision, briefly explaining the Query window, Result window and Item Summary window. It discussed the issues of storing the objects and links in the database so that inter-object connectivity can be provided. That would allow a user to follow hyperlinks to find items interesting to him in the database. Related to the development of the database manager are the issues of the MARIAN search engine. The development of the MARIAN system and the feasibility to port it to Alpha were described.

The following chapter describes the development of the database manager module. It describes the data loading process, document storage and retrieval with the database manager.

Chapter 3

DATABASE MANAGER

This chapter gives an in-depth explanation of the Database Manager for Envision. The database manager forms a separate module in the Envision server that is called by the client surrogate as shown in Figure 1.2. It is responsible for storage and retrieval of records. The current version of the database manager runs on NeXT machines. It relies upon the GNU Database Manager (GDBM) to manage underlying database operations. Functions performed by the database manager for bibliographies and articles are:

1. analysis of SGML records for indexing purposes,
2. identifying authors, titles, subjects and the body of the record,
3. loading the database into our version of MARIAN to prepare inverted files and LINK and STRING databases,
4. storage of records with address and length information, and
5. retrieving a SGML record, parsing the SGML and packaging using the EQRL format for either *profile* or *present* information.

A brief overview of the system architecture is given to explain the role of the database manager in the server. Data analysis and data loading processes are then described to facilitate understanding of inverted files, LINK and STRING databases. That is followed by an explanation of data retrieval and the packaging process.

3.1 System Architecture

As Figure 1.2 shows, the system consists of the server (the Envision intermediary, MARIAN search system and Document Database Manager) and multiple clients connected to it through a communication protocol. Communication between the clients and server and within the server is synchronous. The server listens to a well-known port for any connection requests from the client. Upon establishing a connection with the server, each client is given a set of resources and the client executes as a separate thread [BKL93]. Different threads corresponding to different clients execute concurrently submitting queries, performing searches, retrieving results from the database, or interacting with users. The clients and server communicate via the Envision Query Result Language (EQRL) [Hea94] [Fra94]. The protocol is synchronous and allows the client to submit different types of commands such as *StartSession*, *AskQuery*, *RetrieveDocument* and *EndSession* to which the server replies appropriate information such as *Document identification number*, *Document Title*, *Author(s)* and *Document Format* and so on [Hea94]. The servers within the MARIAN searchers follow the User Interaction Protocol as described in Chapter 2 [FFS93]. Also, as described in Chapter 2, the client software offers an X Window based Motif interface to facilitate submission of queries, previewing result sets and viewing documents.

A typical Envision session starts with the client submitting a *StartSession* query. The server acknowledges the connection request by sending a *StartSessionReply* and establishes the connection. At the same time, the server also assigns a set of resources for the client, and spawns off a client surrogate for the corresponding client [Fra94] [BKL93].

Clients can now perform queries using the *Graphic Query Window*. Once the client submits a query, it synchronously waits for the result set. When the query arrives from the Envision Intermediary to the client surrogate, it is passed to MARIAN searchers which parse the query, identify the key terms for searches and search the database to obtain the result set containing *similar* documents. Searchers continue to perform their function until the documents in the result set show *similarity* below a certain threshold defined by the system.

When the bibliographic identification numbers corresponding to the retrieved documents arrive from MARIAN, the corresponding client surrogate is revived, which then calls the Document Database Manager (DDM) for documents. For an explanation of control flow within the Envision intermediary, please refer to Chapter 4. The interaction within the DDM is not thread-safe since there are shared resources such as parsers and the database records. Concurrent execution of two threads could potentially result in a conflict over any of these resources. Hence, access to the database has to be protected. Only one client surrogate can access the database at any given time and other client surrogates are queued by the system. As the documents are obtained from the DDM, they are packaged into EQRL and returned to the client. The DDM then goes on to service the next client surrogate. Clients present the *profile* information in the *Result Window*. At this point, users can either make a new query or obtain more information about a document of interest [FHN93] [BCF93]. If they choose the latter option, a call is made to DDM to obtain the selected document in its entirety. DDM as usual pulls out the document. However, before packaging it in EQRL, the document is sent to the SGML-to-HTML translator which translates the document on the fly to HTML. The translated document is packaged in EQRL and sent back to the client, which launches NCSA Mosaic with the corresponding document [Lee93] [LCo93]. Note that the document can either be an article, a bibliographic entry, a video segment, or a page image. The documents are *presented* in the most suitable viewer for the document format.

Alternatively, users can perform new queries on the database and retrieve result sets. When the user finally decides to quit, he/she selects the *Quit* option from the *File* menu which sends an *EndSession* command to the server. The server replies to it by terminating the client surrogate, releasing the resources corresponding to the client surrogate, and acknowledging by replying with an *EndSessionReply* [Fra94].

This section briefly explained the system architecture and the control flow within the server. It helped motivate the importance of the DDM in the server. The next two sections explain the data loading and retrieval processes, respectively.

3.2 Database Schema

Before queries can be serviced, data has to be loaded and index files prepared. This section explains the data loading process for Envision. Figure 3.1 shows the different phases in data loading. Loading is generally a separate time-consuming process and can be carried out independently of system development. In the current version, we have about 100,000 SGML records loaded in Envision.

This section explains the database schema, storing terms and links in the database, creation of databases, and inverted files and their relevance in the search or retrieval process [FFo92].

3.2.1 Root and Uncategorized String Database

These databases are at the heart of the data processing and retrieval routines. The root database contains a list of tuples consisting of terms from the *Collins English Dictionary* and the term identifier *WordID* in the database. The uncategorized string database has a similar list of tuples for terms not defined in the *dictionary*. The uncategorized string database is updated with new terms as they are found by the processing routines. Table 3.1 and 3.2 present examples from the root and uncategorized string databases, respectively.

As each term is identified by the Analyzer in Phase I, if it is an English word, the corresponding term identifier is obtained from the root database. Otherwise, the uncategorized string database is searched to determine if the term occurs. If it is not present in the uncategorized string database, a new identifier is assigned to the term and the uncategorized string database is updated [FFo92].

3.3 Processing Steps

For each SGML record, links are formed from an *author* or a *subject* to the document. After all the records in the database are loaded, link databases are created that link all the *authors* and *subjects* in the database with the document identifiers in which they occur.

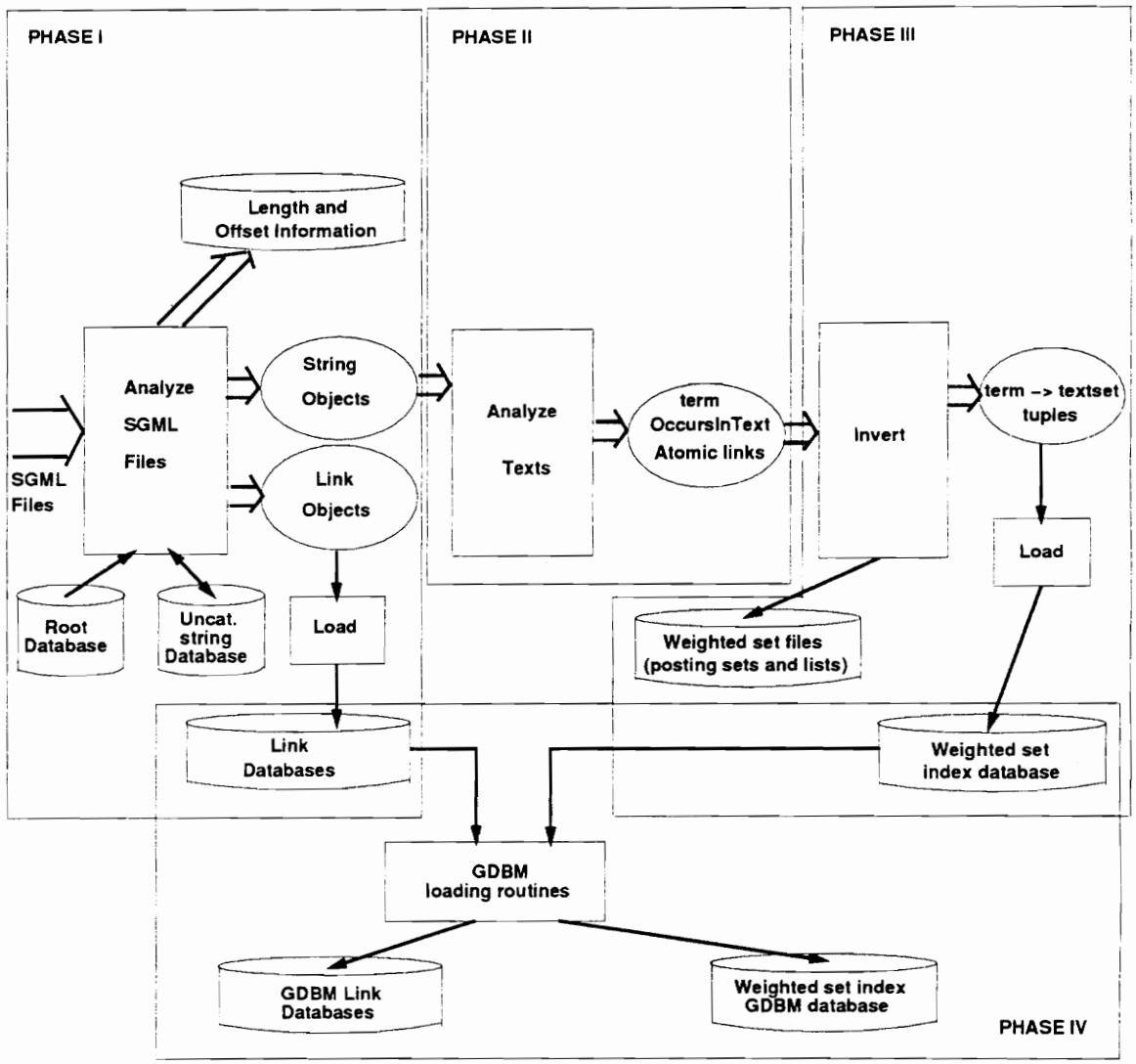

Figure 3.1: Major Processing Steps in Data Loading

Table 3.1: Examples from the Root Database

WordID	Term
...	...
.....	.....
78298	user
.....	.....
19737	algorithm
.....	.....
.....	.....
47603	interfaces
.....	.....
.....	.....
38915	fibre
...	....
.....	.....

Table 3.2: Examples from the Uncategorized String Databases

WordID	Term
..	...
.....	.....
255027	Hartson
.....	.....
.....	.....
264153	Northway
.....	.....
263878	Morarji
.....	.....
470562	Hix
.....	.....
...	....
.....	.....

MARIAN searchers follow these links to identify relevant documents [FFS93].

We also identify and store strings such as the *title* or *body* for each document. Text analysis routines identify the number of occurrences of each term in the document and number of documents the term occurs in the database to determine and assign weights to individual terms. The inverted files are then created with links from terms to the relevant documents. All the databases are finally loaded with the GNU Database Manager.

3.3.1 Phase I: Analysis of SGML Records

As shown in Figure 3.1, the first step is to analyze the SGML records. A typical SGML record is shown in Figure 3.2. However, records can be full-text articles, or bibliographic entries with abstracts [Br93A] [Br93B]. The SGML files are preprocessed to replace any SGML entities with the corresponding ASCII characters before analysis. Hence, the entity `&equals` is replaced by =, and `&divide` is replaced by /. Preprocessing is essential, as MARIAN loading routins can not handle the SGML entities. The analysis process involves extracting the text between various SGML *begin* and *end* tags (a data entity) and categorizing it. The information content of the record is categorized as described below into one of the four categories: *Author*, *Title*, *Subject* and *Body* as required for loading.

As each SGML record is parsed, the Analyzer identifies the data entities to be loaded in the database. A data entity is classified into one of the four categories based on the following definitions:

- Any *author*, *editor* or *artist* in the SGML record falls in the *Author* category.
- The *Title* category constitutes data entities *title*, *subtitle*, *in* and *series*.
- A *Subject* category is defined by *CR Category*, *keywords* and *terms*.
- Anything else that doesn't fall in any of the above categories is identified as content of the *Body*.

<b.jrnart id="HartsonSiochiHix90">
<au> <fnm> H. </fnm> <mnm> Rex </mnm> <snm> Hartson </snm> </au>
<au> <fnm> Antonio </fnm> <mnm> C. </mnm> <snm> Siochi </snm> </au>
<au> <fnm> Deborah </fnm> <snm> Hix </snm> </au>
<title> The UAN <subtitle> A User-Oriented Representation for
Direct Manipulation Interface Designs </subtitle> </title>
<series> Research Contributions </series>
<in> ACM Transactions on Information Systems </in>
<date> <year> 1990 </year> </date>
<volume> 8 </volume> <number> 3 </number> <pp> 181-203 </pp>
<crt> (c) Copyright 1990 Association for Computing Machinery </crt>
<keywords>
<keyword> Software engineering </keyword> <keyword> Requirements </keyword>
<keyword> specifications </keyword> <keyword> Languages </keyword>
<keyword> Software </keyword> <keyword> engineering </keyword>
<keyword> Tools and techniques </keyword> <keyword> User
interfaces </keyword> <keyword> Software engineering </keyword>
<keyword> Design </keyword> <keyword> Representation </keyword>
<keyword> Design </keyword> <keyword> Human factors </keyword>
<keyword> Languages </keyword> <keyword> Behavioral design </keyword>
<keyword> Constructional design </keyword> <keyword> Human-computer
interface </keyword> <keyword> Representation of interfaces </keyword>
<keyword> Task analysis </keyword> <keyword> User interface </keyword>
</keywords>
<abstract> <p> Many existing interface representation techniques,
especially those associated with UIMS, are constructional and focused
on interface implementation, and therefore do not adequately support
a user-centered focus. But it is in the behavioral domain of the user
that interface designers and evaluators do their work. We are seeking
to complement constructional methods by providing a tool-supported
technique capable of specifying the behavioral aspects of an
interactive system-the tasks and the actions a user performs to
accomplish those tasks. In particular, this paper is a practical
introduction to use of the User Action Notation (UAN), a task and
user-oriented notation for behavioral representation of asynchronous,
direct manipulation interface designs. Interfaces are specified in
UAN as a quasihierarchy of asynchronous tasks. At the lower levels,
user actions areassociated with feedback and system state changes.
The notation makes use of visually onomatopoeic symbols and is simple
enough to read with little instruction. UAN is being used by growing
numbers of interface developers and researchers.

```
In addition to its design role, current research is investigating
how UAN can support production and maintenance of code and
documentation. </p> </abstract>
</b.jrnart>
```

Figure 3.2: A Sample SGML Record in the Envision Database

One function of the Analyzer is to assign a bibliographic identification number (bibid) to each document. Bibids are assigned in ascending order starting from 1. The Analyzer initializes a table for each of the four categories defined above [ASU86] [LMB92]. A table descriptor is returned for each of the categories. The Analyzer then calls loading routines with the bibid and the appropriate table descriptors to prepare LINK databases, LINK files and STRING files. These are shown in Table 3.3, Table 3.4 and Table 3.5, respectively [FFo92].

A LINK file is created for categories *Author* and *Subject*. A LINK file maintains links between a *term* and the corresponding SGML record. Hence, as shown in Table 3.4, the *RecordID* represents the identifier for the SGML record and the *WordID* represents the identifier for a *term* (here author). The column *LocationInRecord* is a combination of two subfields *category* and *instance*. The *Author category* is represented by 200. Similarly, the *Subject*, *Title* and *Body* are represented by codes 100, 300 and 400 respectively. The *instance* for *Author* is *a*, *e* or *r* depending on *authorname*, *editorname* or *artistname* in the SGML record. An *e* identifies it to be an editor and *r* identifies an artist. The author database records the links between the *WordID*, and actual *Name* as shown in Table 3.3. Terms in the *Subject* category are similarly treated and Subject database and LINK files for the *Subject* are created.

A STRING file is created for each of the categories *Title* and *Body*. These files are processed further to find the number of occurrences of terms and to assign weights to terms accordingly. As shown in Table 3.5, a STRING file for the *Title* category has the

Table 3.3: Link Database for the Author Category

WordID	Name
...	....
...	....
14946	Lenwood S. Heath
...	....
...	....
9819	H. R. Hartson
...	....
8819	Deborah Hix
...	....
...	....
14839	Edward A. Fox
...	....
.....	.....

Table 3.4: Link Processing for the Author Category (Author.link)

RecordID	WordID	LocationInRecord
....	....	....
11776	8819	200a
....	....	....
14545	8819	200a
....	....	....
16651	14839	200a
16651	14944	200a
16651	14945	200a
16651	14946	200a
....	....	....
....	....	....
17721	9819	200a
17721	16180	200e
17721	8819	200a
..	..	...
....	....	....

Table 3.5: String Processing for the Title Category (Title.string)

RecordID	TextString
...	.....
...	.....
16651	Order Preserving Minimal Perfect Hash Functions and Information Retrieval
...	.....
17721	A User-Oriented Representation for Direct Manipulation Interface Designs; The UAN
...	.....
...	.....
87105	Algorithmic program debugging
...	.....
.....	.....
108036	Abstraction mechanisms in hypertext
.....	.....

record identifier *RecordID* and a corresponding *TextString*. The STRING file for the *Body* is similarly created. Note that if an SGML record has a *title* and *subtitle*, the STRING file would contain subtitle concatenated to the title.

The LINK and STRING files created in this step are used further to create indices as explained in the next section.

Storing Address and Length Information

Each SGML file contains a number of SGML bibliographic entries (also referred to as records). As each file containing SGML records is processed, the address of each record within the file and length of the record (in number of bytes) is stored. This helps in efficiently retrieving the record from the file.

A *master* file maintains a list of tuples in ascending order. Each tuple is composed of two elements: the last record identifier *(Last RecordID)* for a file and the address of the file *(Location Pointer)* containing those SGML records. A selection from the *master index file* for Envision is shown in Table 3.6. Note that the tuples are maintained in ascending order of *Last RecordIDs*.

Table 3.6: Master Index File (Master.out)

Last RecordID	Location Pointer
.....	...
.....	...
16635	BIBS/hcibib/IR89.sgml.pre
16663	BIBS/hcibib/IR90.sgml.pre
16699	BIBS/hcibib/IR91.sgml.pre
.....	...
17713	BIBS/hcibib/TOIS07.sgml.pre
17728	BIBS/hcibib/TOIS08.sgml.pre
.....	...
.....	...
43055	BIBS/vadisc1a/vlsifa.sgml.pre
.....	...
.....	...
108093	CACM/MAC.in/89May628SametWoody.done.pre
.....	...

The difference between the *Last RecordIDs* on two adjacent lines gives the number of SGML records at the address pointed to by the latter *Last RecordID*. In Table 3.6 the last record in file *BIBS/hcibib/IR89.sgml.pre* has bibid 16635 and the last record in file *BIBS/hcibib/IR90.sgml.pre* has bibid 16663. This means that the number of SGML records in file *BIBS/hcibib/IR90.sgml.pre* is 16663 – 16635 = 28. Similarly, since *Last RecordID* for file *BIBS/hcibib/IR91.sgml.pre* is 16699, and the master file is maintained in ascending order of *Last RecordIDs*, number of records in file *BIBS/hcibib/IR91.sgml.pre* is 16699 – 16663 = 36.

For each file containing SGML records, as shown in Table 3.7, a corresponding file containing address and length information is prepared. This table contains information about the *RecordID*, its address within the file defined by the *Byte Offset* and *Length* of the record. So, for the Table 3.7, the SGML record with *RecordID* = 17720 would start at a byte offset 17014 within the file and the length of the record is 1999 bytes. Continuing, the next record would start at byte offset 17014 + 1999 = 19013. Its length is 2663.

Table 3.7: Address and Length Information

RecordID	Byte Offset	Length (in bytes)
....	..	....
....	..	....
....	..	....
17717	9253	2465
17718	11718	3131
17719	14849	2165
17720	17014	1999
17721	19013	2663
17722	21676	2905
....	..	....
.....	...	.....
.....	...	.....

The Analyzer routines can be executed as follows:

```
analyzer -b <bibid file> -f <address file> <  <SGML file>  >  <parsed file>
```

where `<bibid file>` is the file that stores the current bibid, `<address file>` is the file name for address and length information, `<SGML file>` is the input file containing SGML records, and `<parsed file>` is the output file generated by the Analyzer.

3.3.2 Phase II: Occurrences of Terms within the Database

LINK and STRING files and LINK databases created in the analysis step are used further to determine weights of individual *terms* in the database. Several factors such as the number of occurrences of the *term* within a document and the number of documents the *term* occurs in determine the weight to be given to the *term* [FFo92].

Tables 3.8 and 3.9 show a sample of the *Occursinauthor.dat* and *Occursintitle.dat* files. During the creation of these files, the databases for *root* and *uncategorized strings* are accessed. A class identifier is assigned to each of the two databases. Class 101 corresponds to the *root database* and class 2 corresponds to the *uncategorized string database*. The field

Table 3.8: Author Occurrences in Documents (OccursinAuthor.dat)

FullTermID	RecordID	# Occur	Offset,Size	SubFieldText
..	..	...	...	..
101:1707	27307	ONE	<1,1>	B. Vidal
..	..	...	...	...
2:470562	8819	ONE	<9,3>	Hix
..	..	...	...	...
..	..	...	...	..
2:255027	9819	ONE	<8,7>	H. Rex Hartson
..	..	...	...	..
2:255027	56304	ONE	<7,7>	H. R. Hartson
..	..	...	...	..
....	....	.....	.....	..
101:4997	41795	ONE	<1,3>	Don Fussell
....	....	.....	.....	..
....	....	.....	.....	..

FullTermID defines a tuple consisting of the *class identifier* and the *term identifier*. The corresponding *record identifier* is defined in the second field. Values in the third column can either be ONE, FEW or MANY. A *term* is defined to occur FEW times if its number of occurrences is greater than 1 and no greater than some upper limit, currently 20,000. A *term* occurring just once has ONE in the #Occur column. All other terms are defined to be MANY. Offset and size define the offset of the first letter of the *term* within the field and size gives the number of bytes for the *term* [FFS93] [FFo92].

Typically, the occursin routines are executed as follows:

```
occurs_in -c <  <database_name>  >  <database_name.occursin>
```

where option -c signifies normal text. < `Database_name` > can be of any of the four categories *author*, *title*, *subject* or *body*.

After creating < *database_name.occursin* > file, it is sorted to obtain < *sorteddatabase_name.occursin* > files as follows:

```
SORT.4 <  <database_name.occursin>  >  <sorted database_name.occursin>
```

Table 3.9: Title Occurrences in Documents (OccursinTitle.dat)

FullTermID	RecordID	# Occur	Offset,Size	SubFieldText
....	...	..	....	...
....	...	..	....	...
101:78298	17721	ONE	<2, 4>	user
101:78298	17751	ONE	<29, 4>	user
....	...	..	....	...
....	...	..	....	...
101:19737	38409	ONE	<3,9>	algorithm
....	...	..	....	...
101:47603	1896	ONE	<27,9>	interface
.....	.....	....	.....	....

3.3.3 Phase III: Inverted Files

The final step is to create an inverted file with links to a document if the *term* occurs just ONCE in the database, or to a set of documents if the term occurs FEW times in the database. The example inverted files are shown in Tables 3.10 and 3.11. The *FullTermID* defines the class of the term (such as a root term or an uncategorized string) and the term identifier. The *Occur Rate* defines the rate of occurrence of the term in the database based on the # Occurrences. If # Occurrences is 1, the *SrcRecID/PlistPtr* defines the bibid, otherwise it defines the pointer to the posting list file. The *Weight/PlistLength* defines either the term weight or the length of the posting list for the term, depending on the # *Occurrences*. As in the *occursin* files, the tables are indexed by the *FullTermID*. The *Occur Rate* is 1 if the *term* occurs just ONCE in the database and the next field then defines the *RecordID*. However, if the *term* occurs FEW times in the database, the rate of occurrence of the *term* is given and an address in the posting list file is given where the sequence of documents for that term can be found. The last column defines the *weight* for the term in the database [FFo92].

Inverted files are created as follows:

```
invert -f <sorted database_name.occursin> -d <database_name.occursin>
```

Table 3.10: Inverted File for the Author Category (Author.inv)

FullTermID	Occur Rate	#Occurrences	SrcRecID/PlistPtr	Weight/PlistLength
...	....	...	..	...
...	....	...	..	...
2:255027	0.903536	2	457552	3
...	....	...	..	...
...	....	...	..	...
2:470562	1	1	8819	0.9
...	....	...	..	...
...	....	...	..	...
101:542	0.939138	2	667368	2
...	....	...	..	...
101:81460	1	1	79108	0.9
...	....	...	..	...
.....	.....	....	....	.....

Table 3.11: Inverted File for the Title Category (Title.inv)

FullTermID	Occur Rate	# Occurrences	SrcRecID/PlistPtr	Weight/PlistLength
....	..	..	...	...
....	..	..	...	...
2:31464	1	1	19243	0.65901
....	..	..	...	...
2:37588	1	1	56155	0.774648
....	..	..	...	...
....	..	..	...	...
101:78298	0.350302	2	6358328	1858
....	..	..	...	...
....	..	..	...	...
101:19737	0.247723	2	1065680	6098
....	..	..	...	...
.....	...	....	....	....

```
-o <database_name.inv>
```

3.3.4 Phase IV: Loading with GNU Database Manager

Finally, all the databases are loaded with the GNU database manager (GDBM). Source and sink files for *author* and *subject* categories are prepared to point to and from the *bibid* to the *termID*. Thus from the *termID*, we can find the document in which the *term* occurs. GDM utilities are then used to access these links and retrieve the document identifiers.

Data can be loaded incrementally as long as any of the previously loaded records are not modified or deleted. There is no limitation of number of documents that could be added incrementally. However, as described in Appendix B, the Analyzer should be able to update the *existing databases* for authors and subjects. If the Analyzer does not find the databases for authors and subjects in the current working directory, it would create new databases for authors and subjects. That is an undesirable situation since important indexing data is lost. Hence, Analyzer and author and subject databases should be in the same working directory. However, as there is no means to propogate any modification or deletion of data to the generation of indexes, loss or modification of data results in fatal errors and the only solution in such cases is to reload.

3.4 Retrieval of Data

This section explains the retrieval of records from the database, and packaging the records in EQRL to be sent back to the client.

When a user submits a query to search for documents, depending on the field being searched for, the query parts are handled by their respective parsers. Author parsers, for example, look up in their own database to determine the *term identifier(s)* for author name(s). The parser then searches the LINK database to follow the links for different *term identifier(s)* to obtain a list of relevant documents. For text parsers which handle titles, the query terms are first looked up in the *root* and *uncategorized string* databases to find their

respective term identifiers in the database. The inverted files are then searched to access the pointers to the relevant documents. In cases of multiple search terms, MARIAN uses a weighted heuristic search to return the result set.

As bibids are retrieved by the MARIAN-Envision BackEnd, they are passed to the document database for retrieving the records. The retrieved SGML records are stored as a temporary file in the server. This process is essentially the same irrespective of whether the client asked for the *profile* or the *present* information.

A typical retrieval process then would be to search in the *master index* file for the SGML file for the given bibliographic identification number (bibid). Then, search the *address and length table* for the byte offset and length for that bibid. Once the byte offset and length for the record are known, an *fseek* call with appropriate parameters sets the file pointer at the desired location and pulls out the required number of bytes.

Thus, for example, if a user submits a query to retrieve documents for author name *Hix*, the search for corresponding documents is executed as follows:

Root and *uncategorized* string databases are searched for term *Hix*. As shown in Table 3.2, the *WordID* for *Hix* is 470562. The class identifier for uncategorized strings is 2. Hence the *FullTermID* for *Hix* is 2 : 470562. Now, from Table 3.10, we can see that the # of occurrences of *Hix* is just 1. Note from Table 3.8, the # of occurrences for *Hartson* is at least 2. In that case, a pointer to the posting list file is obtained from Table 3.10 which would point to the list of *termIDs* such as 9819, 56304 for *Hartson*. The *SrcRecID* for 2 : 470562 is 8819. Hence, now we look in Table 3.4 for document identifiers *(RecordIDs)* corresponding to the *termID* 8819. We find the matching *RecordIDs* 11776, 14545 and 17721. The DDM is then called with a list of *RecordIDs*.

Now, for the *RecordID* 17721, we search in Table 3.6 for the *file name* for the *RecordID*. It can be seen that *RecordID* 17721 should occur in file *BIBS/hcibib/TOIS08.sgml.pre* as explained earlier. The address and length information for the corresponding file is shown in Table 3.7. Hence, the disk pointer is positioned at address 19013 for file *BIBS/hcibib/TOIS08.sgml.pre* and 2663 bytes corresponding to *RecordID* 17721 are retrieved and stored. Figure 3.3

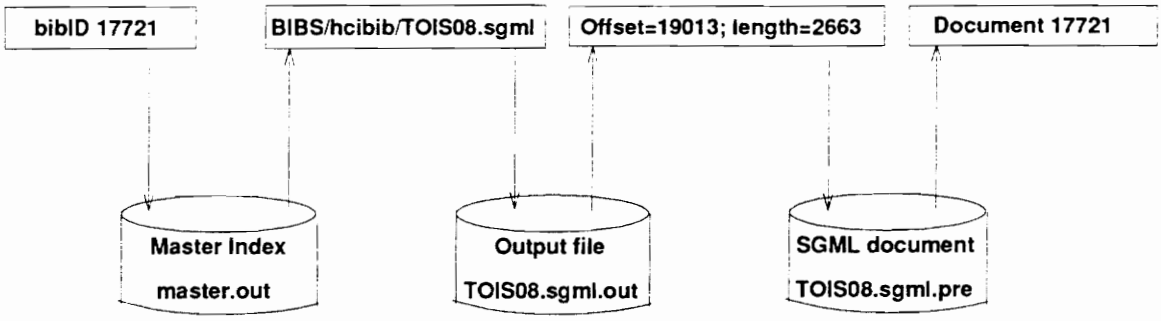

Figure 3.3: Retrieval Process for bibID 17721

shows diagrammatical representation of the retrieval process. This process is carried out for all the *RecordIDs* retrieved. A *Profile* of each document is then prepared and sent to the client as explained next.

3.4.1 Profile

If the clients ask for *profile* information, the profile of each record is sent back. A profile contains the important information about the record such as *author*, *editor*, *title*, *crcategory*, *year*, *document relevance*, *document format* and *document identifier*. Parsers are used to extract this information from the SGML record and suitably package it in EQRL to be sent back to the client. The client is sent back a profile for each bibid retrieved. A sample EQRL translated profile for the SGML record in Figure 3.2 is shown in Figure 3.4.

3.4.2 Present

If the user is interested in the full-text of a document or wants to view the document, he asks the server to *present* the document. The bibid of the document (obtained from the *profile*) is sent by the client to the server. The server then calls the document database with the bibid asking for the whole document. The document database retrieves the document as explained above. However, an SGML document has to be translated to HTML before it could be presented to the user. Hence the SGMLtoHTML translator is called to translate the document to HTML. The document database adds the HTML translated document to the

```
DOCUMENTID:"17721"
RELEVANCE:760
AUTHOR:"Hartson, H. ";
"Siochi, Antonio ";
"Hix, Deborah "
TITLE:"The UAN; A User-Oriented Representation for Direct Manipulation
Interface Designs"
YEAR:"1990"
CRCATEGORY:""
FORMAT:HTML
DOCUMENTSIZE:0
TIMESCITED:0
DOCUMENTCITES:0
JOURNAL:"ACM Transactions on Information Systems"
ABSTRACT:""
```

Figure 3.4: EQRL for *Profile*

EQRL for *present*. EQRL allows information similar to that for *profile* except that it adds information about the *DocumentFileName* on the client side and the *most suitable viewer* for the current document. The EQRL representation for the SGML record in Figure 3.2 is shown in Figure 3.5.

For documents not in SGML, for example a video document, a Uniform Resource Locator (URL) [Lee93] [LCo93] pointing to the document is sent back to the client with other information such as *SessionID*, *QueryID*, *Format*, *Application*, etc. The client then launches the most suitable viewer for the document with the help of NCSA Mosaic and presents the document.

3.5 Summary

This chapter explained the functions of the database manager – the data loading process, creation of inverted files, storing and retrieving data, packaging the data for *profile* and *presenting* information.

```
RESULT
SESSIONID: "772954847"
QUERYID: "2"
FILENAME: "772954847.2.result"
RESULTTYPE: DOCUMENT
STATUS: OK
DOCUMENTFILENAME: "772954847.2.result.HTML"
FORMAT: HTML
APPLICATION: "Mosaic"
BODY: 280
<p>
Birnbrauer, Herman.
(1990)
Acquisition of Skills for New Plant Start-Up in Singapore Along with a
Plan to Retain Employees.
In <I>Proceedings of the Human Factors Society 34th Annual Meeting</I>.
pp. 824-826.
<p>
<P><B>Abstract: </B><P>
<blockquote>N R</blockquote>
<p>
```

Figure 3.5: EQRL for *Present*

Chapter 4

CONCLUSION AND FUTURE WORK

In this project, the role of the database manager in the development of the Envision server were emphasized. Various issues related to it were discussed. The accomplishments of the project are as follows:

1. Analyzing the SGML records

   The test database for Envision which consists of about 100,000 documents including articles and bibliographies was put together. Appendix A contains detailed statistics about them. A preprocessor to translate the *SGML entities* to the ASCII character equivalent was written. Two parsers were written for extracting entities from these records as explained in Chapter 3 [LMB92] [ASU86].

2. Data Indexing

   Bibliographic entries and articles were indexed and inverted files for them were prepared. An elaborate description of the different data processing steps was given in Chapter 3.

3. Data Retrieval

   Profile and *Present* functions for the database manager were implemented to retrieve and display an SGML record. Data retrieval functions include extracting out an appropriate number of bytes from the SGML file in the database, parsing the record to extract information, packaging it in EQRL and sending it back to the client [Hea94].

Following is a list of possible extensions to this project, that would enhance the capabilities of Envision or add new features to it.

Issues Related to Data:

- The current database can further be enhanced to add significantly more data formats such as page images, audio, video, PostScript and PDF files. Addition of a document in any of the above formats would require the corresponding SGML record for indexing purposes. Addition of the above data formats would require the client to have a corresponding document viewer on its system.

- One of the drawbacks of the data indexing is the requirement of reindexing all the SGML documents in case of update or when deleting any of the indexed records. It is essential to be able to dynamically load the SGML records.

- An issue related to this project is to be able to store objects in the database and link them. This would allow users to visualize Envision as a database of objects such as authors, institutes, conferences, journals, etc. besides the documents. Users then can retrieve interesting items by walking through the links.

- In the future, as more documents are added to the database, it would be effective to have an automated mechanism for translating input documents that are not in SGML to their SGML equivalent. A language, Description Language for Textual Objects (DELTO), has been defined to accomplish this goal. More information about it can be obtained from [AHe94].

Appendix A

Envision Data Specifications

Data being used in Envision includes publications of the ACM and other literature - in the broadest sense - that can be processed for loading and use. A number of resources are being used, including a large quantity of literature of various types:

1. Bibliographies
   - a collection of bibliographies from University of Utah, University of Arizona
   - bibliographies from ACM's Transactions on Software Engineering and Methodology,
   - SIGGRAPH bibliographies,
   - HCI bibliographies from Ohio State University,
   - bibliographies from University of Melbourne (VA Disc 1 bibliographies), and
   - bibliographic entries including *Guide to the Computing Literature*
2. Full-Text Articles covering *Communications of the ACM* (CACM) and from Design Automation conferences and newsletters;
3. Multi-Media Documents

All the bibliographies listed above were either in *BibTex*, *Refer* or *SGML* format.

Table A.1 provides information about the Envision documents that are loaded into the database.

Table A.2 lists the source and SGML versions of the available bibliographies. All the source files are on *video.cs.vt.edu* in directory tree */u1/envision*. The corresponding SGML files are on *fir.cs.vt.edu* in directory tree */envdata2/dalalk/pre*.

Table A.1: Document Statistics

Source	Original Format	#Records	Collection Size (Kbytes)	Average Record Size(bytes)	#Authors
Bibliographies					
University of Utah	BibTex, SGML	11095	4029	363	18391
Caches	BibTex	62	40	645	121
Ecoop	Refer	26	34	1307	83
HCI Bibliographies	Refer, SGML	7182	8239	1147	14661
SIGGRAPH Bibliographies	BibTex, SGML	15588	6682	428	31771
Network Flow Problems and STOC-FOCS	BibTex, SGML	2779	1217	437	5148
ACM Transactions on Software Engineering and Methodology	BibTex, SGML	21	36	1714	38
University of Arizona	BibTex, SGML	4797	2615	545	8697
VA DISC1 Articles	SGML	1548	848	547	4867
VA DISC1 Bibliographies	SGML	590	253	428	980
VA DISC1 Journals	SGML	27170	10480	386	44048
VA DISC1 Periodicals	SGML	13185	6832	518	30833
VA DISC1 Reports	SGML	2821	1773	628	4793
VA DISC1 Tech. Reports	SGML	242	138	570	243
VA DISC1 Miscellaneous	SGML	3427	1480	431	6149
GUIDE to Computing Literature	GUIDE	17497	15110	860	28289
Full Text Articles					
CACM	ASCII Text	64	2177	34051	185
Multi-Media Documents					
Video		1	495	495	2

Table A.2: Locations of Source and SGML Versions of Documents

Document	Source	SGML
Bibliographies		
University of Utah	BIBS/beebe	BIBS/beebe
Caches	BIBS/caches	BIBS/cache
Ecoop	BIBS/ecoop	BIBS/ecoop
HCI Bibliographies	BIBS/hcibib	BIBS/hcibib
SIGGRAPH Bibliographies	BIBS/siggraph	BIBS/siggraph
Network Flow Problems and STOC-FOCS	BIBS/stocfocs	BIBS/stocfocs
ACM Transactions on Software Engineering and Methodology	BIBS/tosem	BIBS/tosem
University of Arizona	BIBS/uarizona	BIBS/uarizona
VA DISC1 Articles	BIBS/vadisc1/a	BIBS/vadisc1a
VA DISC1 Bibliographies	BIBS/vadisc1/b	BIBS/vadisc1b
VA DISC1 Journals	BIBS/vadisc1/j	BIBS/vadisc1j
VA DISC1 Periodicals	BIBS/vadisc1/p	BIBS/vadisc1p
VA DISC1 Reports	BIBS/vadisc1/r	BIBS/vadisc1r
VA DISC1 Tech. Reports	BIBS/vadisc1/t	BIBS/vadisc1t
VA DISC1 Miscellaneous	BIBS/vadisc1/x	BIBS/vadisc1x
GUIDE to Computing Literature	GUIDE/xaa – GUIDE/xbs	GUIDE/xaa – GUIDE/xbs
Full Text Articles		
CACM	CACM/MAC.in	CACM/MAC.in
Multi-Media Documents		
Video		VIDEO/misc

Appendix B

Data Loading Procedure

B.1 Introduction

Raw Envision data files are stored in directory

`/envdata1/dalalk/data`

The preprocessed data files are stored in directory

`/envdata1/dalalk/pre`

The indexes prepared by the data loading process for the second load are stored in directory

`/envdata2/dalalk/load2/index`

on DEC Alpha *fir.cs.vt.edu*. The directory structure for the data files and the indexes correspond to each other. The source code, the scripts and the utilities for the data loading procedure are on NeXT *opac1.cc.vt.edu*. These utilities will be described later in detail for each individual phases. All the utilities and programs discussed in this appendix are NeXT binaries. For load II, data loading was performed on *opac1.cc.vt.edu*.

B.2 Phase I

For each file loaded in Envision, several files containing index information are prepared. An explanation of file extensions for index files in data loading is as follows:

- *.pre*

  Preprocessed data files

- *.author*

  Author files prepared by phase I of data loading. These files contain links from *termID*

to *bibID*. They also contain additional information about the *termID* as to whether the term corresponds to an *author*, *editor* or *artist*. Table B.1 describes the codes assigned to categories and subcategories in data loading.

- *.subject*

  Subjects files prepared by phase I of data loading. These are similar to author files in their structure. Subject files represent links from a subject to the bibliographic entry. As in the author files, subject files consist of links from the *termID* to *keywords*, *terms* or *CRcategory* names in the bibliographic entry.

- *.title*

  Title files prepared by phase I of data loading. They contain a list of tuples where each tuple is composed of the *bibID* and the title of the corresponding bibliographic entry. All the fields in the *Title* category such as *title*, *subtitle*, *in* and *series* are concatenated and stored in the title file.

- *.body*

  Body files prepared by phase I of data loading. They are similar in structure to title files. All the terms that are not categorized in either of the above three categories are considered to be a part of the body.

- *.parse*

  Parse files generated by the Analyzer. These are used mostly for debugging purposes. They contain the extracted data entities marked-up between SGML tags in the bibliographic entry as they are parsed from the data files, but before they are actually loaded. The parse files are a checkpoint to make sure that the data entities are loaded in correct format and are valid.

- *.out*

  Output files which contain address and length information about all the Envision data files (data files have extension *.prc*). There is a one-to-one mapping between each data

Table B.1: Category and Subcategory Codes for Envision Data

Category	Category Code	Subcategory	Subcategory Code
Title	100	Series	e
		In	i
		Subtitle	s
		Title	t
Author	200	Author	a
		Editor	e
		Artist	r
Subject	300	CRCategory	c
		Keywords	k
		Terms	t
Body	400	Address	a
		ISBN	b
		Copyright	c
		Date	d
		Edition	e
		Language	g
		ISSN	h
		In Chapter	i
		Code Number	j
		Library of Congress	k
		Length	l
		Medium	m
		Number	n
		Note	o
		Page Numbers	p
		Frequency	f
		Price	r
		Section Number	s
		Rating	t
		Volume	v
		Abstract	z

file and the corresponding address file. As described in Chapter 3, these files contain a list of tuples where each tuple consists of the *bibID*, the starting byte offset of the bibliographic entry and the length of the bibliographic entry.

The data loading process uses the root and uncategorized string databases. The locations of these databases are as follows:

Root Database: `/big/marian/databases/root/root`; Uncategorized String Database: `/Virginia/marian/data/string/unCatStr.fixed`

The scripts and executables for data loading are in directory

`/users/dalalk/envision/parser`

on *opac1.cc.vt.edu*. A description of contents of this directory is as follows:

- preprocessor

  Subdirectory that contains the preprocessor that replaces SGML entities by equivalent ASCII characters. The executable is *preprocess* and the source code is in file *preprocess.l*.

- bibliparser

  Subdirectory that contains scripts, source code and executable for loading bibliographies. The script *phase1.bibli* calls the executable *analyzer.bibli* to analyze and load bibliographies. The source code for the Analyzer is in the file *parser_bib.l*

- articleparser

  Subdirectory that contains scripts, source code and executable for loading articles. The script *phase1.article* calls the executable *analyzer.article* to analyze and load articles. The source code for the Analyzer is in the file *parser_art.l*

- loaddatabase

  This subdirectory contains the loading script *load*. This script creates a directory structure for target directory where indexes for data files in source subdirectories would be stored. It then calls either *phase1.bibli* or *phase1.article* as appropriate.

B.2.1 The Analyzer

In the following discussion, for the sake of generality, *phase1.** are used to refer to either *phase1.bibli* or *phase1.article*. *Analyzer.** is used similarly.

The *phase1.** script executes the *analyzer.** which starts the data loading process. As explained above, for each file that is loaded *.author*, *.title*, *.subject*, *.body*, *.out* and *.parse* files are prepared. These files are stored in the corresponding subdirectory under the target directory tree. Also, the *phase1.** script prints a line to stdout for each file as it is loaded to give feedback to the user. Since there are several directories, each of which contains several other subdirectories, it is a good idea to break up phase I for each subdirectory to establish a checkpoint. At that point, several tests can be performed to make sure about the integrity and validity of data loading. For example, in the directory `/envdata2/dalalk/pre` on *fir.cs.vt.edu* there are four subdirectories *BIBS*, *GUIDE*, *CACM* and *VIDEO*. Starting with BIBS which contains about 75000 bibliographic records, it is a worthwhile effort to make sure that they all are loaded correctly before proceeding further.

*Analyzer.** creates author and subject databases during its execution. It also keeps track of the number of records loaded at each point in its execution. Hence, it is important that *analyzer.** is run from the same directory all the time. Moving the Analyzer across directories can result in loss of important data.

At the end of phase I of loading you should have the following:

1. Database files *db_Author.pag* and *db_Subject.pag* created in the same directory as the Analyzer.

2. Index files **.author*, **.title*, **.subject* and **.body* created in the target directory tree with correspondng path for each file in the source directory.

3. Address and length file **.out* created in the target directory with the corresponding path for each file in the source directory.

4. Record file *record* created in the same directory as the Analyzer.

The databases created in phase I can be listed by using utility *db_dbmp.old* in directory

`/Virginia/envision/bin`

on *opac1.cc.vt.edu* as follows:

```
db_dump.old <pag file name (without extension)> > <database name>
e.g.; db_dump.old db_Author > Author.load2
```

where `<pag file name>` is either *db_Author.pag* or *db_Subject.pag*. These files dump a list of tuples where each tuple is composed of two entities; a term ID and the actual term where term is either an author name or the subject. These databases can always be used to verify the validity of the data in the load. The loaded data is verified by confirming the *term ID* for the author or the subject in these databases with those in the LINK files.

B.3 Phase II

In phase II, *.dat* and *.sort* files are created. These files are stored in the directory

`/envdata2/dalalk/load2/occursin`

on *fir.cs.vt.edu*. The *.dat* files are created as a result of execution of *occurs_in*. The *.sort* files are the sorted occurs_in and are prepared by *SORT.4* as explained below.

The utility *occurs_in* for phase II is in directory

`/Virginia/envision/bin`

on opac1.cc.vt.edu and is used as follows:

```
occurs_in -c < <input database name> > <output occursin file>
```

where -c stands for normal text.

Occurs_in files are prepared for all four categories: author, title, body and string.

The input files for occurs_in for author and subject category are the databases created in phase I by using *dbdump.old*. For title and body categories, input files are prepared by concatenating all the *.title* and *.body* files created in phase I. These files do not necessarily have to be concatenated in the order in which they were created.

The sorted occurs_in files are prepared by using utility *SORT.4*. This is a filter and is used as follows:

```
SORT.4 < <input occurs_in file> >  <sorted occurs_in file>
```

At the end of phase II, you should have occurs_in and sorted occurs_in files for all the four categories. It is a good idea to start a script while doing this phase so as to keep a log of activities occurred during the execution of occurs_in. The scripts created during the loading of phase II are also stored in directory
`/envdata2/dalalk/load2/occursin`.

B.4 Phase III

Phase III creates inverted files which are prepared by using utility *invert*.
An explanation of files created by this phase is as follows:

- *.inv*

  These are the inverted files containing links from the *termID* to the *bibID* or to a *.plist* or *.pset* file.

- *.plist*

  These are posting list files. If the number of occurrences of a *termID* is FEW in the database then the inverted file contains a pointer to posting list file where the *bibIDs* for each *termID* in the posting list are stored.

- *.pset*

  These are posting set files which store information for a *termID* whose number of occurrences in the database are MANY. Similar to the posting list files, these files contain a set of *termIDs* and their corresponding *bibIDs*.

As in the previous phase, all of the above files for load II are stored in directory
`/envdata2/dalalk/load2/occursin`
on *fir.cs.vt.edu* and are created as follows:

```
invert -f <sorted occursin file> -d <occursin file> -o <output invert file>
```

As in the previous phase, this utility should be executed for all of the four categories of data.

At the end of phase III, you should have inverted, posting list and posting set files for all the categories of data.

B.5 Phase IV

Finally, the inverted files are loaded with the GNU Database Manager (GDBM). GDBM output files have two different extensions. Files with the *.gdbm* extension are the GDBM files. Files with an *.info* extension are the *information* files containing the total number of documents loaded and the bytes used. These files for load II are also stored in directory

`/envdata2/dalalk/load2/occursin`

on *fir.cs.vt.edu*. These files are created with the help of utilities *inv_util* and *link_util* as follows:

```
inv_util -c <database name>
```

where `<database name>` is the name of the *.inv* file. This would create a GDBM database for the inverted files. As in the previous phases, the *.gdbm* files corresponding to inverted file databases are created for all of the four categories of data. The GDBM databases created for load2 are stored in directory

/envdata1/dalalk/databases/x_inv

on *fir.cs.vt.edu*. The databases thus created are then input to GDBM by utility *gdm_util*. To execute *gdm_util* type

```
gdm_util
```

at the command prompt. Upon entering the *gdm_util*, there are several options. Choose the option to add a database. It will ask for several things such as machine name and path name. Enter appropriate information. When you are finished with it, exit the utility.

For author and subject databases, link databases which store the links from the *termID* to *bibID* are also created. These are created using *link_util* as:

```
link_util -l <database name>
```

where database name is the author database or subject database obtained from phase I by using *dbdump.old* utility. This would create link databases in GDBM format. As for inversion database, link databases also should be made known to GDBM as described above.

This finishes our description of the data loading process for Envision. Now, MEBE can use these databases to perform search and to retrieve bibIDs.

B.6 Retrieval code

The source code and executables for data retrieval are in directory `/usr/home/envision/server-source/doc_db` on *jingluo.cs.vt.edu*. A description of the source code files is as follows:

- *propre.c* It implements the *profile* and *present* calls. Depending on the user requests, an appropriate function is executed.
- *record.c* This function implements functions to retrieve an SGML record from the database given the *bibID*.
- *parse_rec.l* Lex file to parse the record to prepare the *profile* of a document.
- *oneresult.c* After parsing the record, profile information in EQRL is prepared by the functions in this file.

A *master.out* file is also prepared and stored in the directory where *env_interm_server* resides. This file is the master index from which the document database obtains information about the pathname of the SGML file for a document. An example master.out file is shown in Table 3.6. It stores last bibID for each SGML file with the SGML file name. It is important that the *master.out* file be strictly in the order in which data was loaded.

During the retrieval process, the document database creates two temporary files: *profile.out* and *present.out*. These files are created by *record.c* and they store the document most currently retrieved by the document database. *Parser_rec.l* uses these files to parse the document retrieved and prepare an EQRL equivalent of those.

REFERENCES

[AHe94] Guillermo A. Averboch and Lenwood S. Heath, DELTO: Description Language for Textual Objects, *Technical Report*, Department of Computer Science, Virginia Polytechnic Institute & State University, Blacksburg, VA 24061-0106, 1994.

[ASU86] A. V. Aho, R. Sethi and J. D. Ullman, *Compilers: Principles, Techniques and Tools*, Addison-Wesley Publishing Company, Reading, MA, 1986.

[BCo91] Len Bass and Joelle Coutaz, *Developing Software for the User Interface*, Addison-Wesley Publishing Company, Reading, MA, 1991.

[Br93A] Dennis J. Brueni, *Article DTD: Project Envision - A User-Centered Database from the Computer Science Literature*, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061-0106, March 1993.

[Br93B] Dennis J. Brueni, *Project Envision Report on Bibliographies*, Internal Document, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061-0106, March 1993.

[BCF93] Dennis J. Brueni, Bailey T. Cross, Edward A. Fox, Lenwood S. Heath, Deborah Hix, Lucy T. Nowell and William C. Wake, What If There Were Desktop Access to the Computer Science Literature?, In *Proceedings of the 21st Annual ACM Computer Science Conference: CSC'93*, pp. 15-22, Indianapolis, IN, Feb 1993. ACM Press, New York, NY.

[BKL93] Joseph Boykin, David Kirschen, Alan Langerman and Susan LoVerso, *Programming under Mach*, Open Systems Series, Reading, MA, 1993.

[BOS91] Paul Butterworth, Allen Otis and Jacob Stein, The GemStone Object Database Management System, *Communications of the ACM*, 34(10), pp. 64-77, October 1991.

[Che92] QiFan Chen, *An Object-Oriented Database System for Efficient Information Retrieval Applications*, PhD dissertation, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061-0106, 1992.

[Cli93] Ben Cline, *User Interaction Protocol - Implementation Guide*, MARIAN Internal Document, Computing Center, Virginia Polytechnic Institute & State University, Blacksburg, VA 24061-0106, August 1993.

[Com91] Douglas E. Comer, Internetworking with TCP/IP: Principles, Protocols and Architecture, Volume I, 2nd edition, *Prentice Hall Inc.*, Englewood Cliffs, NJ, USA, 1991.

REFERENCES

[Dat87] C. J. Date, An Introduction to Database Systems, Volume I, 4th edition, *Addison-Wesley Publishing Company*, Reading, MA, 1987.

[D^+91] O. Deux et al., The O_2 System, *Communications of the ACM*, 34(10), pp. 34-48, October 1991.

[FCF91] Edward A. Fox, QiFan Chen and Robert K. France, A general reference model for hypermedia and information retrieval and its implementation in CODER / LEND, In *Hypertext / Hypermedia Handbook*, McGraw Hill Inc., New York, 1991, pp. 329-355.

[FFK91] Edward A. Fox, Robert K. France, M. Prabhakar Koushik, Jenny-Lou Menezes, Qifan Chen, Amjad Daoud and J. Terry Nutter, CODER: A Retrieval and Hypertext System Using SGML and a Lexicon, In *Proceedings of ACH/ALLC*, Arizona State University, Tempe, AZ, March 1991, pp. 159-163.

[FFo92] Robert K. France and Edward A. Fox, *Indexing Large Collections of Small Text Records for Ranked Retrieval*, Computing Center and Department of Computer Science, Virginia Polytechnic Institute & State University, Blacksburg, VA 24061-0106, 1993.

[FFS93] Edward A. Fox, Robert K. France, Eskinder Sahle, Amjad Daoud and Ben Cline, Development of a Modern OPAC: From REVTOLC to MARIAN, In *Proceedings of the 16th International Conference on Research & Development in Information Retrieval*, pp. 248-259, Pittsburgh, PA, June 1993.

[FHH91] Edward A. Fox, Lenwood S. Heath and Deborah Hix, *A User Centered Database from the Computer Science Literature*, Project Proposal, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061-0106, 1991.

[Fox92] Edward A. Fox, Building a User-centered Database from the ACM Literature, In *Proc. Symposium on Document Analysis and Information Retrieval*, Las Vegas, Nevada, March 1992, pp. 235-246.

[FHN93] Edward A. Fox, Deborah Hix, Lucy T. Nowell, Dennis J. Brueni, William C. Wake, Lenwood S. Heath and Durgesh Rao, Users, User Interfaces, and Objects: Envision, a Digital Library, In *Journal of the American Society for Information Science*, 44(8), September 1993, pp. 480-491.

[Fra92] Robert K. France, *MARIAN: Detailed Design*, MARIAN Internal Document, Computing Center, Virginia Polytechnic Institute & State University, Blacksburg, VA 24061-0106, August 1992.

[Fra94] Robert K. France, *Envision Back-End Design Narrative*, Envision Internal Document, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061-0106, March 1994.

REFERENCES

[GAA94] Henry M. Gladney, Zahid Ahmed, Ron Ashany, Nicholas J. Belkin, Edward A. Fox and Maria Zemankova, Digital Library: Gross Structure and Requirements, *IBM Research Report #RJ9840*, IBM Almaden Research Center, San Jose, California, May 1994.

[Gol90] C. F. Goldfarb, The SGML Handbook, *Clarendon Press*, Oxford, 1990.

[Hea94] Lenwood S. Heath, *Envision Query/Result Language*, Envision Internal Document, Department of Computer Science, Virginia Polytechnic Institute and State University, Blacksburg, VA 24061-0106, March 1994.

[Her90] Eric Van Herwijnen, Practical SGML, *Kluwer Academic Publishers*, Boston, USA, 1990.

[HHa93] Deborah Hix and H. Rex Hartson, Developing User Interfaces, John Wiley & Sons Inc., USA, 1993.

[Lee93] Tim Berners-Lee, *HyperText Transfer Protocol*, Internet Draft, IIIR Working Group, CERN, Switzerland, November, 1993.

[LCo93] Tim Berners-Lee and Dan Connolly, *HyperText Markup Language*, Internet Draft, IIIR Working Group, CERN, Switzerland, June 1993.

[LMB92] John R. Levine, Tony Mason and Doug Brown, Lex and Yacc, *O'Reilly & Associates Inc.*, Sebastopol, CA, 1992.

[Rao93] Durgesh Rao, *A Domain Model of Computer Science for an Information System*, Internal Document, Department of Computer Science, Virginia Polytechnic Institute & State University, Blacksburg, VA 24061-0106, 1993.

[RMa87] Francis Rhys and Ian Mathieson, A Threaded Programming Environment, In *Proceedings of the 10th Australian Computer Science Conference*, Geelong, Australia, February 1987, pp. 173-183.

[Sah93] Eskinder Sahle, *Development of a User Interface for MARIAN and CODER Systems*, Master's Project Report, Department of Computer Science, Virginia Polytechnic Institute & State University, Blacksburg, VA 24061-0106, 1993.

[Shn92] Ben Shneiderman, Designing the User Interface, 2nd edition, *Addison-Wesley Publishing Company*, Reading, MA, 1992.

[Ste92] W. Richard Stevens, UNIX Network Programming, *Prentice-Hall of India Private Ltd.*, New Delhi 110 001, India, October 1992.

[TRG87] Avadis Tevanian, Jr., Richard F. Rashid, David B. Golub, David L. Black, Eric Cooper and Michael W. Young, Mach Threads and the Unix Kernel, In *Proceedings of the USENIX 1987 Summer Conference*, Phoenix, AZ, June 1987, pp.185-197.

VITA

Kaushal Dalal, son of Rameshchandra and Heena Dalal, was born on 25th September 1971 in Pune, India. He completed his secondary school from Seth M. A. High School in Bombay. Later he completed his high school from Parle College in 1988. He then joined the prestigious Victoria Jubilee Technical Institute (V.J.T.I.) affiliated with the University of Bombay for a Bachelor's degree in Computer Engineering, obtained in July 1992. It was at V.J.T.I. that he learnt the basics of computers and developed an incipient interest in computer science. As a testimony of which, he joined Virginia Tech in Fall 1992 for a Master's in Computer Science.

He is interested in sports, enjoys music and loves to travel and explore the world.