Development of a Modern OPAC: From REVTOLC to MARIAN*

Edward A. Fox†#, Robert K. France#,
Eskinder Sahle†, Amjad Daoud†and Ben E. Cline#

Department of Computer Science†and Computing Center#
Virginia Polytechnic Institute and State University
Blacksburg VA 24061-0106

Abstract

Since 1986 we have investigated the problems and possibilities of applying modern information retrieval methods to large online public access library catalogs (OPACs). In the Retrieval Experiment — Virginia Tech OnLine Catalog (REVTOLC) study we carried out a large pilot test in 1987 and a larger, controlled investigation in 1990, with 216 users and roughly 500,000 MARC records. Results indicated that a forms-based interface coupled with vector and relevance feedback retrieval methods would be well received. Recent efforts developing the Multiple Access and Retrieval of Information with ANnotations (MARIAN) system have involved use of a specially developed object-oriented DBMS, construction of a client running under NeXTSTEP, programming of a distributed server with a thread assigned to each user session to increase concurrency on a small network of NeXTs, refinement of algorithms to use objects and stopping rules for greater efficiency, usability testing and iterative interface refinement.

*This work was funded in part by grants from the National Science Foundation (Grants IRI-8703580 and IRI-9116991), CLR (Grant 4048-C), OCLC, PRC, and the Virginia Center for Innovative Technology.

1 Introduction

In 1986, the Virginia Tech Library provided us with tapes containing all MARC records from their online public access catalog (OPAC) system, which uses software supported by VTLS Inc. Since that time a coordinated research and development program focused on OPACs has been underway, involving the Department of Computer Science, the Computing Center, and the Library. This paper summarizes our activities and findings, and the resulting software and systems.

Important aspects of our work include:

- working with large (3-900,000 record) collections;
- controlled experimental studies with many test subjects on the use of advanced retrieval methods for large OPACs;
- research with parallel processors (Sequent), and development of a distributed, client-server production system on a small network of low-cost workstations (NeXTs), with threaded processing under MACH;
- use of a specialized object-oriented DBMS and object-oriented programming languages;
- application of principles of dialogue independence and usability testing of the interface;
- providing an experimental OPAC search system as an alternative to the University Library's regular OPAC system;
- morphological analysis using an online lexicon;

- optimization of space through compression and a special inverted file organization;
- optimization of processing through stopping rules for search; and
- simulation of user workload and monitoring of system performance to support further tuning.

The sections below provide details regarding our (largely unpublished) efforts. Section 2 summarizes the experimental studies (REVTOLC), which prompted our subsequent development work. Section 3 briefly explains the object-oriented DBMS (LEND) that we have developed and which underlies the current production system (MARIAN). Section 4 gives details of the design, development, and user testing of MARIAN. Section 5 discusses our plans for future work, and Section 6 acknowledges many of those who have assisted.

2 REVTOLC

In 1986 the Virginia Tech Library was considering adding Boolean search capabilities to its VTLS system, and so dumped all of the MARC records onto magnetic tapes in preparation for inversion on a mainframe computer. We were given a copy of the tapes to use for experimentation, and so carried out the Retrieval Experiment — Virginia Tech OnLine Catalog (REVTOLC) study.

2.1 Problem, Opportunity, and Approach

Many of the world's libraries depend in part, or exclusively, on an online public access catalog (OPAC) so that patrons can locate desired publications, thus eliminating use of conventional card catalogs [23]. Typically, expensive, centralized, mainframe computers are involved. These systems allow the public to use computers to search for information about various resources like books, serials, and audio-visual publications [20]. However, OPACs are not always very efficient, effective, or usable. This is in part due to the distinctive characteristics of the problem area. OPACs must service a heterogeneous group of chronically infrequent users with complex relationships between goals and behaviors and difficult information retrieval tasks [1]. Some typical deficiencies of current systems include: difficulty in expanding on query results, excessive retrieval and null retrieval, inconvenient displays, and under-utilization of system features [31]. These serious problems threaten our ability to support universal access to information.

Happily, advances in computer technology, information retrieval, and human-computer interaction provide an exciting opportunity to explore and solve these important problems. In particular,

- advances in computer technology allow us to develop low-cost, highly efficient, modular systems;
- advances in information retrieval facilitate improvements in system effectiveness (along with query expansion and avoidance of excessive retrieval); and
- advances in human-computer interaction can empower diverse patrons by providing highly usable systems (with convenient displays and ready access to appropriate features).

Our approach has been to take advantage of these opportunities through a coordinated program of research and development. We conducted two controlled experiments, in 1987 and 1990, to determine which advanced retrieval methods would be most useful for searching OPACs. As discussed in Sections 2.2 and 2.3, we also explored interface techniques that would work well with those retrieval methods. Further, as we proceeded with the experiments, we became aware of numerous aspects of system development necessary for a large-scale implementation of our findings; many of these are discussed in Sections 3 and 4.

2.2 1987 Study

The first REVTOLC study was carried out during the Spring semester of 1987. The key features of our pilot experiment are:

- comparison of vector, vector feedback, Boolean, and extended Boolean (i.e., p-norm [12, 28]) retrieval methods;
- volunteer assistance by 52 freshman English class students;
- searching against approximately 300,000 records; and

- use of the 1985 version of the SMART system, enhanced with our own code, including methods for fast inverted file creation [18], running on a VAX 11/785 under ULTRIX.

For more details see [14].

This hurried experiment was flawed in a number of ways, prompting us to work on a more carefully designed study. While it seemed clear that users liked the vector methods, we lacked important data necessary to analyze timing or effectiveness. Further, controls relating to the interface were weak, clearly creating a bias against the Boolean-type methods. Finally, it became clear that more users, more questions, and more records would be needed to give adequate statistical power and generalizability to our findings.

2.3 1990 Study

Supported in part by grants from CLR and OCLC Inc., we began work on a more careful experiment, assisted by Linda Wilson of the Library. Numerous library staff helped by:

- collecting questions from patrons, from which our set of 18 was selected;
- carrying out repeated searches, and providing expert judgments on those and on the searches of the experimental subjects, to allow us to measure relative recall; and
- advising on the development of the experimental system and its testing.

A brief summary of some of our early efforts on this project appears in [15].

We compared the same four retrieval methods considered in the 1987 study, but in most other aspects this was a larger investigation, as can be seen in Table 1.

In all cases our interface operated on VT100 terminals or terminals that emulated that type of behavior. The display was organized into areas so that users only needed to fill in forms. A simple, specially designed editor was available for entering or changing data in any of the screen areas.

We hypothesized that advanced retrieval methods would be preferred and more effective than the conventional Boolean approach, that p-norm would be preferred to Boolean, and that the vector methods would be particularly easy to use.

Table 1: Key Characteristics of 1990 Study

Characteristic	Value
Number of users	216
Number of documents	500,000
Number of questions	18
Retrieval methods	4
Retrieval methods/user	2
Searches/user	4
Search system	SMART (extended)
Computer system	Sequent (10 nodes)
Operating system	Dynix (UNIX)

2.3.1 Experimental Design

The user interface was carefully controlled so that the user experienced minimal differences while using the several retrieval methods. Written instructions, an online tutorial, and practice exercises all served to facilitate a fair comparison of the retrieval methods. Further, we tried to balance system performance so that retrieval time was approximately the same for each method.

By giving each user up to 90 minutes for the experiment, and only paying them their $5 fee after they had completed all tasks, we tried to ensure that everyone was properly motivated and not hurried. Most users finished soon after 60 minutes, though some liked the system so much that they stayed an extra hour to use it with their own questions.

Each user was randomly assigned to one of 12 test groups, and worked with two methods, searching using each method for two of the 18 test questions. These groups allowed us to eliminate the effects of the order of the two methods. Further, by having each user search for records for each of four questions, using two different methods, we were able to measure intra-subject differences as well as between-subject effects, adding to the power of the study.

To ensure generality of our results we balanced the subjects with respect to gender, academic level (freshman to senior to graduate student), and college. We also collected the demographic data shown in Table 2 by online questionnaire.

We gathered a great deal of other data for subsequent analysis. In particular, we logged each session

Table 2: Demographic Data Collected

No. of computer courses taken
Frequency of use of computers
Frequency of use of Virginia Tech OPAC
Frequency of use of other retrieval systems
Typing skills
Gender
User patience
Overall grade point average
Academic level
Academic area

(with timestamps), recorded retrieved documents and those marked by the users as relevant, and collected user comments in online questionnaires.

2.3.2 Results

Our analysis to date has only considered the online questionnaires, and so primarily reflects user perceptions. We grouped all users together, since none of the demographic factors listed in Table 2 was found significant at the $p = 0.05$ level.

The most important (significant) findings are:

- Users generally felt that the documentation was satisfactory and that the online tutorials provided sufficient introduction to the retrieval methods.
- Users found it easy to formulate queries from the given questions, for all retrieval methods.
- Users felt that methods that ranked the results gave higher precision.
- Users felt that p-norm searching was best in terms of finding more of the relevant documents in a single search iteration while producing less "noise." In this regard, users also preferred the two vector methods to Boolean.
- Users felt that vector searching with relevance feedback was best in terms of finding a larger total number of useful documents over all iterations. Users also preferred p-norm to Boolean.
- Users felt that the easiest method to use was vector with feedback, with Boolean the most difficult. P-norm was easier than Boolean.
- Users felt that vector with feedback took the least time to arrive at a satisfactory set of results, with Boolean taking the most. The same was found regarding being an effective aid for casual search.
- Users preferred the vector methods to the Boolean and p-norm methods regarding: effectiveness for comprehensive searching, ease of learning, and ease of use.
- Users found the REVTOLC system easier to use than Virginia Tech's current OPAC system.

Further details regarding this experiment can be found in [11]. Additional analysis of the data collected is pending availability of adequate funding.

Our results seem to indicate that the type of interface provided, and the vector with feedback retrieval method, would be well received by OPAC users. Subsequent sections describe the consequent development of a production system including these features.

3 LEND

While working with REVTOLC, we learned a great deal regarding requirements for a production system:

- With large numbers of records, there are numerous situations in which efficient hashing methods would be helpful (e.g., to find a string in the "dictionary," to locate a record given its identifier, to find an entry in an authority file, or to look up a call number).
- Many types of "objects" are involved, and can be efficiently described using inheritance.
- Caching and other methods that reduce disk access lead to faster performance.
- Efficient management of data in memory and on secondary storage devices is important, and requires a significant amount of code.

These and other needs helped motivate work on the Large External object-oriented Network Database (LEND) system [7], which satisfies all of the above requirements. First, LEND supports building of minimal perfect hash functions that are used internally to optimize performance, and that can be built with

user data to guarantee minimum disk accesses and near minimal space overhead. Second, LEND is an object-oriented DBMS, that supports a class hierarchy with inheritance, so that all of the classes of objects needed can be quickly defined, without unneeded repetition of code. Third, LEND's lowest or storage layer supports objects in main memory, in page sets, or in UNIX files. An LRU class allows reasonable cache performance, a buffer class speeds access to small objects, and page and UNIX file classes simplify efficient accessing of disks. Fourth, LEND's object layer, operating atop and hiding the details of the storage layer, supports efficient hash indexing, caching, and buffering. Primitive classes include integer, real, and string. Composite classes include set, tuple, and list. Data access classes include several based on hashing, plus AVL trees for in-memory data. Finally, LEND implements much of the **information graph** model we have proposed [7]. Thus, it allows us to support traditional DBMS, IR, and knowledge-base processing through operations on a graph of objects. LEND has classes for both nodes and arcs, includes several types of interfaces to support various views and sets of operations, and has a query language allowing retrieval in terms of nodes, arcs, paths, and graphs.

The second version of LEND was completed in Spring 1992, using g++, and is being ported to "vanilla" C++, removing dependence on the GNU class system. LEND has been licensed by several organizations, and has been thoroughly tested and used in the MARIAN OPAC system, discussed in the next section.

4 MARIAN

The Multiple Access and Retrieval of Information with ANnotations (MARIAN) system has been under development at Virginia Tech since 1991. Our aim was in part to rectify mechanical and conceptual problems [4] of OPACs with techniques like [25]: morphology-based matching, query expansion, authority files, linking and terminological aids and direct interfaces. We hoped to avoid command-line style interaction and to have a system that could be easily used with minimal training [24]. Our approach was not only to enhance the interface but also to redesign the underlying system, going beyond Boolean

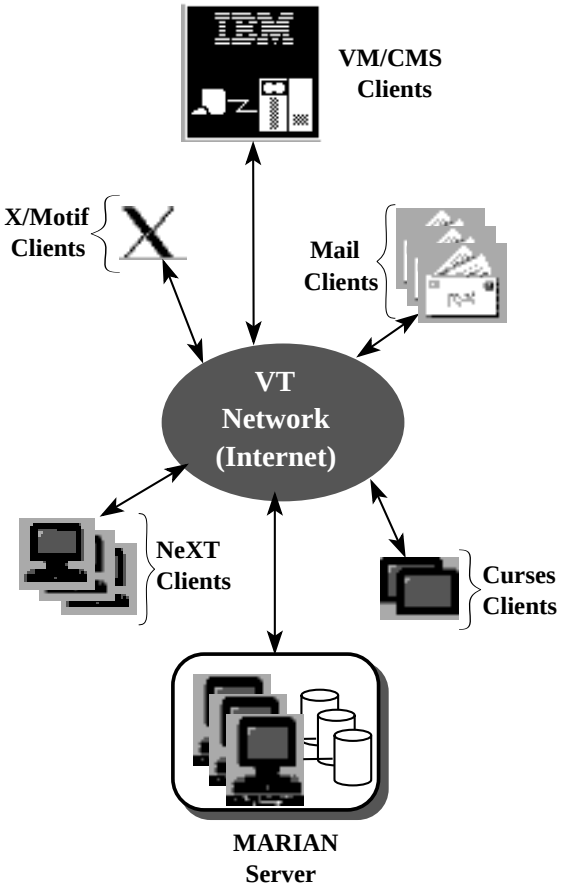

Figure 1: Client/Server with Multiple Interfaces.

queries [21]. The long term goal is to progress toward a truly "intelligent" system with broad usability and "smart" functionalities [22], building upon our prior work with the COmposite Document Expert/extended/effective Retrieval (CODER) testbed [13].

From the perspective of users, MARIAN is a central server, holding library catalog data, as shown in Figure 1. For those working on a NeXT or on a workstation running X/Motif, a local client will handle their interaction with the server. For those connected via terminals to VM/CMS or a UNIX system supporting CURSES terminal control, specialized clients interconnect the terminals with the server. Finally, for any system supporting mail, interaction with MARIAN will be possible using that mode of communication.

To understand MARIAN in more detail, it is best to follow an example of its use, as given in the next subsection.

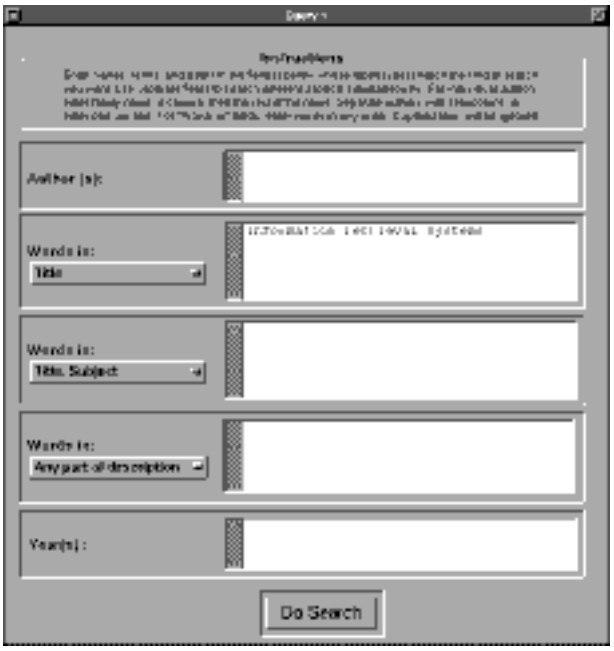

Figure 2: Query Entry.

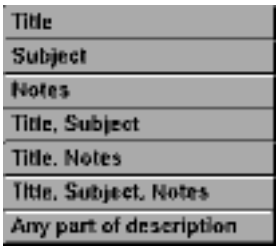

Figure 3: Pop-up for Field Selection.

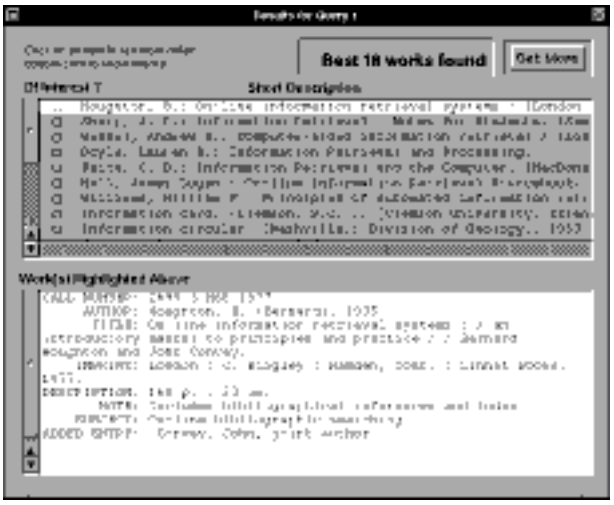

Figure 4: Presentation of Results.

Figure 5: Query History.

4.1 Example

Figures 2-6 illustrate current operations of MARIAN, using screen dumps from a NeXT system with the present version of our client software. Interaction begins with a menu and optional login screen, designed to provide password security as necessary. After a user selects *New Query* from the menu, a query can be entered using a form-style window, as shown in Figure 2.

Because users often want to search for the same terms in several parts of the MARC records, a pop-up selection panel is provided for query entry, as shown in Figure 3.

Once the server locates, ranks, decompresses, and sends back the desired number of records, these results are shown in a new window, as in Figure 4. In the top pane a user can mark an item as relevant, or select it, which causes it to be highlighted in that pane and displayed in the bottom pane.

Users can work with any number of queries, submitting new ones before results have appeared, in a flexible mixed-initiative style of interaction [16]. Therefore, it is important that a history of queries is maintained, like that shown in Figure 5. This feature supports users asking to edit and/or resubmit previously prepared queries.

The interface also supports various types of feedback searching, that will be handled by server code to be completed early in 1993. The menu options shown in Figure 6 illustrate that a user can ask for other documents like the one that is currently selected, can request standard relevance feedback from all documents selected as relevant, or can look for more items (with lower ranks) retrieved by the current query.

Now that the basic operation of MARIAN has been illustrated, further subsections provide details of the design and implementation.

Figure 6: Initiating a Feedback Search.

4.2 Objects

To understand the operation of the object-oriented MARIAN system, it is best to consider the types of objects involved. This fits in well with our use of LEND, and of the C++ and Objective-C languages.

Many of the objects managed by MARIAN are persistent. These are handled by LEND over the long-term, but are processed by the rest of the system as needed.

4.2.1 Persistent Objects

There are six main types of persistent objects: (1) annotation, (2) text component, (3) authority, (4) document, (5) link, and (6) inverted file entry. The first, not yet in use, will support annotation of the catalog records. Short text blocks, i.e., collections of strings, will be attached by library staff, faculty, and students (with suitable editorial controls). We expect useful messages, critiques, recommendations, and comments that will help future searchers.

The second includes standard strings and variants, as well as various classes of numbers and numeric codes. Standard strings include words found as lemmata in the *Collins English Dictionary*; variants include regular inflections and derivations computed on the fly, as well as irregular forms derived from the same source. These are particularly helpful for the morphological processing carried out during document and query indexing. Other strings not reducible to words in the dictionary are added as they appear in new documents.

The third type of information involves authoritative or canonical representations.

Subject fields in library records are drawn from the controlled vocabulary of the *Library of Congress Subject Headings* (LCSH). These headings are arranged in a complex network including both hierarchical and non-hierarchical relations as well as modifications involving place, time, and content descriptors. Personal and corporate names are likewise controlled. Normalization of names is particularly important so that matches can be identified, and partial matching must reliably handle problems with abbreviations, initials, titles, multi-part surnames, hyphenation, and organization types. In both these cases, identification of the objects satisfying a user query involves more than matching text fields.

Fourth are the documents themselves, which we preserve in their entirety with a lossless compression based on Huffman coding. For the catalog records we decompress into the standard *MARC tape format* using ASCII/ANSEL characters, then reformat into the user's choice of human-readable display formats. Where the user's workstation or terminal permits, clients map ANSEL characters to and from the local expanded character set.

Fifth, in keeping with the philosophy of information graph modeling, there are databases of links between documents and either names or subject headings (the *hasAuthor* and *hasSubject* relations). This representation allows the system to map in either direction between MARC records and authors or LCSH subjects.

Finally, for efficient retrieval, we have inverted files for each text field in the subject, name, and MARC record objects. Each has inverted index entries that include weights, and hashing functions that support fast access to each entry. We optimize both space and processing time by having three subclasses: ONE for entries that occur in a single document, FEW for entries that appear as a standard list of postings, and MANY for terms so common that it is only effective to search them in combination. Posting lists for FEW terms are stored in non-increasing order by weight. MANY terms are organized into a lattice of posting lists where each node corresponds to a combination of MANY terms.

4.2.2 Internal Objects

Since the MARIAN server has been coded largely in C++, and the NeXTSTEP client is in C and Objective C, there are many internal objects. Among the most important objects are: (1) choice, (2) query, (3) retrieved list, and (4) document list.

Throughout an interaction with MARIAN, user decisions are represented by *choice* objects that include both the string and object identifier forms. Also from users, *query* objects are obtained, structured in conjunctive normal form so that complex Boolean as well as vector expressions can be considered. The basic constituents of these objects are any of the various types of terms (e.g., English words, author names, LCSH subjects) and their weights.

As the result of searching, a *retrieved list* is prepared, that has three parts for each entry. First is the identifier of the document to be retrieved. Second is an estimate of its relevance; the list is ordered in descending order of these values. Third is another object that gives the evidence for this presumed match — typically it indicates locations of query terms in the document. Note that after a search, the retrieved list is reduced to a simple list of document identifiers, in the same order, for display. These *document list* objects are also useful in other contexts.

4.3 Architecture and Protocols

While a great deal of the design of MARIAN is illustrated in the above discussion, key aspects relate to the system architecture and decisions regarding communication protocols. Our objective of having a low-cost, high efficiency system that could be easily enhanced and incrementally grown to support increases in data and transactions argued against the approach taken in the REVTOLC study. In particular, having clients communicate with a server running the SMART system [17] was not appropriate.

Rather, we sought an architecture with multiple threads (at least one in each functional module for each user transaction) to provide increased parallelism, and that allowed processes to be distributed as efficiency dictates across a network of workstations. NeXT computers seemed a good choice for initial implementation, since the underlying MACH operating system supports efficient message-based interprocess communication on one or a collection of workstations, as well as threads within a given module. The MACH Interface Generator (MIG), supported on NeXT and OSF/1 systems, is a program that generates remote procedure calls (RPCs) for efficient communication between processes and (relatively) easy interchange of high level data structures [3]. MIG calls are used for all communication inside the MARIAN server irrespective of the nodes on which the linked modules are running.

Communication between clients and the MARIAN server follows the User Interaction Protocol (UIP) developed at Virginia Tech, supported by a locally developed subroutine package [8]. UIP consists of two layers. The upper layer corresponds to the user interface objects described in Section 4.2.2. The lower layer is a symmetrical, remote procedure call protocol for the transport of user interface objects between the MARIAN server and its clients. Data encoding is provided by a combination of Sun's eXternal Data Representation (XDR) [9] and UIP specific data encoding procedures. The protocol runs as an application layer protocol over TCP/IP. The implementation of UIP is thread-safe for support of concurrency in the MARIAN server. A version of UIP that does not require threads is available for clients running on platforms that do not support threads. We decided not to use Z39.50 until it achieves a higher level of functionality and the many proposed changes and extensions to it are completed.

Our initial plan was to use two NeXT computers as the main server, and to run CURSES-based clients on a third NeXT. Since the CURSES package is not thread-safe, each VT100-type terminal connecting would require a separate process, encouraging separation of that load from the searching and other activities.

5 Server Design

The MARIAN server has been carefully designed [16] to be extensible and to ultimately become a third-generation OPAC search system [22]. Figure 7 illustrates the server's three layers: Interface Management, Access Method, and Database Management. Dotted boxes indicate parts whose implementation has not yet been completed. Arrows indicate the direction of MIG calls between modules.

Details on some of the most important modules are given below:

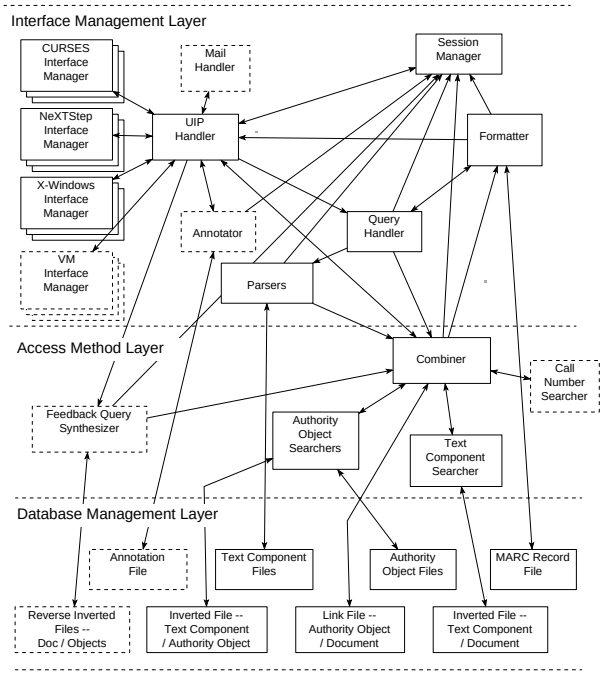

Figure 7: Detailed Design for MARIAN.

- The *UIP Handler* is the interface between the various user interface managers and the rest of the system. Thus it helps enforce the principle of dialogue independence [19], allowing the MARIAN server to function without regard to the different types of clients. It does this by implementing the hierarchy of *interaction objects* defined in the UIP protocol (see Section 4.3). The UIP Handler keeps track of the network location of all active clients, and maps calls for user interaction originating within the server to UIP messages to objects owned by the correct client. By the same token, it translates UIP messages originated by a user action on an interaction object into MIG calls on the server module that created the object.

- The *Session Manager* handles user login, identification, access control, and accounting. It oversees the progress of all currently active user sessions, and of the major activities within each session. It is responsible for much of the error handling, and for graceful termination of internal processing associated with a stopped session. Session status is maintained and can be queried, so that "smart" clients can detect problems or unforeseen delays. Other MARIAN modules report their progress at key points of processing. Thus, the Session Manager brokers all help requests, filling most from its understanding of the current session context. It is the site for any future user modeling enhancements. The Session Manager also maintains preference data for each user across sessions, and may at some point allow sessions to be saved and restored.

- The *Annotator*, when completed, will act as a note manager for the MARC records, with entries linked as for hypertext. It will allow users with proper authorization to add, edit, or examine annotations.

- The *Query Handler* is responsible for initiating all search and browse events. In the latter case it helps the user choose the point at which browsing should begin, manages the browsing process, and in the event that something is identified to be used in a query, passes it to the appropriate parser. In the former, the Query Handler creates a *query* interaction object of the appropriate class, labels it, and sends it to the user via the UIP Handler. (Figure 2 shows a *biblioFormQuery* object as realized by the NeXTSTEP client.) For each query submitted by the user, messages are sent simultaneously to the appropriate parser(s) for each field, as well as to the Combiner so it may coordinate the search. For feedback queries, the Feedback Query Synthesizer is invoked. Finally, the Query Handler keeps track of the query history of each session, and allows users to select, edit, and resubmit old queries.

- The various *Parsers* translate user's representations, usually sequences of characters, to system representations, usually objects that are part of the query object (see Section 4.2.2). They are specialized to the field and type of data involved:

  - Text, the default, assumes English words, and is composed as shown in Figure 8. It reduces running text to a linear combination (or *term vector)* of text component IDs.
  - Date identifies single dates or date ranges.

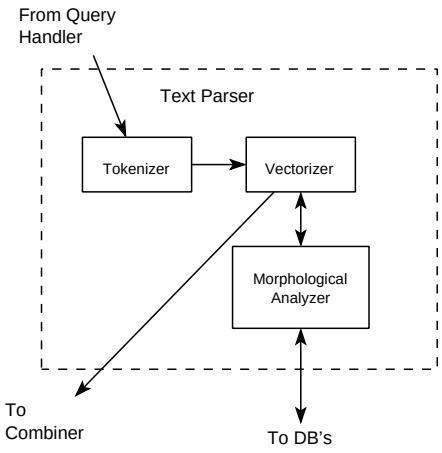

Figure 8: Text Parser.

  - Subject produces both a term vector and a list indicating order of term occurrence, for (partial) matching against the LCSH authority entries.
  - Author separates each author, and prepares a sequence of representations like those used by the Subject Parser.

- The various *Searchers* reply to the Combiner, returning either a retrieved set (see Section 4.2.2), a portion thereof, or an estimate of its size. Similar processing takes place for searchers involving both Text Components and Authority Objects. Usually the k best items are needed, and are found using a specially developed heuristic scheme for *frontier exploration* aimed at stopping the search as early as possible, under the assumptions that weights are used and that similarity can be expressed as a sum of partial similarities:

  1. Terms with inverted file entries of class ONE have their single documents added directly to the candidate list.
  2. Each term with class FEW entries has its posting list scheduled for exploration.
  3. Terms with class MANY entries are merged into a single posting list for joint exploration.
  4. A frontier is established across the collection of posting lists, such that all postings above the frontier are at least a certain weight, and all below have less.
  5. An associative *accumulator* stores those documents found in the explored region above the current frontier, keeping track of the k documents with highest partial similarity.
  6. If the top k documents in the accumulator have stabilized, they are returned and the exploration terminates. Otherwise, the frontier is extended and step 5 is repeated.

- The *Combiner* is at the center of query processing. It coordinates the search process, and can schedule Searcher(s) to perform various types of search. When partially specified authority or high frequency text objects have too many matches, it can request help from the user through the UIP handler. Results go to the Formatter.

- The *Formatter* receives from the Combiner a single document identifier, a document list (see Section 4.2.2), or an addition to a previous list. These go to the user via the UIP Handler, for subsequent selection and display.

The Database Management Layer of MARIAN deals with the various persistent objects discussed in Section 4.2.1. It relies upon the services of LEND, as discussed in Section 3. Further details, especially regarding performance, can be provided in the final version of this paper.

5.1 Interface Design

Because of ease of development and testing, the first MARIAN client has been developed using the NeXTSTEP environment [26]. While most coding in MARIAN has been in C++, interface-related routines followed the NeXTSTEP standard of using Objective C [10]. The object-oriented approach for NeXTSTEP interface development fits in well with the MARIAN philosophy. In particular, MARIAN's client/server communication using the UIP protocol involves *interaction objects*, some of which are shown in Figure 9.

The NeXTSTEP client/server pair can be viewed in terms of the ISO/OSI model of communication (Figure 10). On the client side, the User Interface

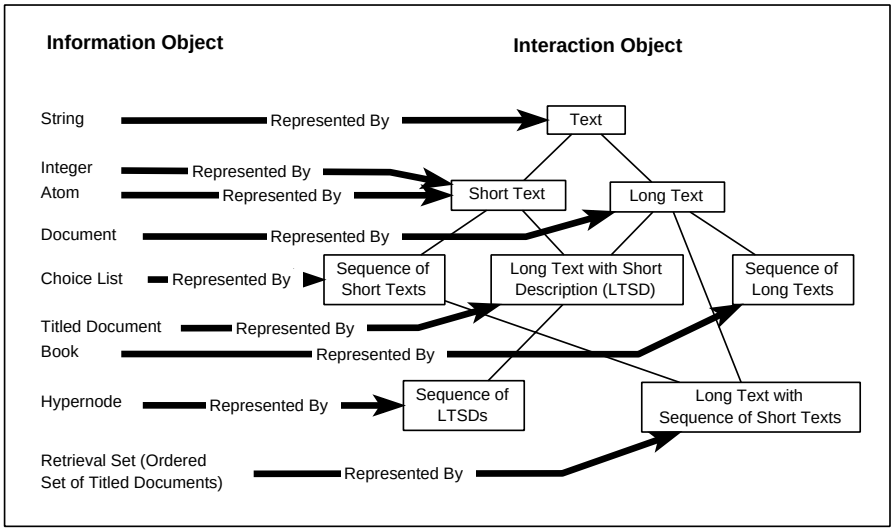

Figure 9: Representation of Internal Information Objects by User-Manipulable Interaction Objects.

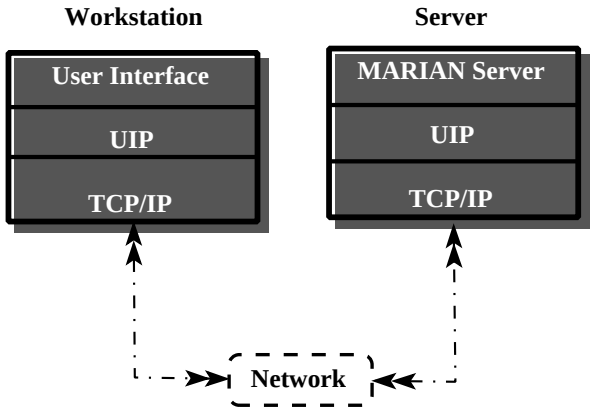

Figure 10: Protocol Stack.

and UIP layers are the most important part of the Application Layer, while on the server side the UIP layer is at the "bottom" of the Application Layer.

The overall client/server processing on NeXT can be seen more clearly in Figure 11. Here the roles of TCP/IP, UIP, and MIG calls for communication should be clear. For further details on the MARIAN interface the reader is referred to [27].

5.2 User Testing

Development of the NeXTSTEP client proceeded in parallel with the MARIAN server. Normal testing procedures have been followed, to ensure correct behavior in terms of specifications. Also, we are committed to iterative refinement and careful usability testing.

Consequently, in October 1992 a carefully selected group of five individuals served for initial usability testing, carried out in concert with another related research effort (Project Envision [5]). The most important shortcomings and additional requirements were:

- Provide more feedback, especially notifying the user regarding: successful invocation of MARIAN, search status after query submission, and server failure.
- Reorganize menu options: renaming to better express actions accomplished, and having hierarchy reflect task organization.
- Provide better information on how the system operates, e.g., explaining the ranking principle.
- Provide more control to users, e.g., maintain user settings and selections regarding window sizes, and format/order of results.
- Simplify the manner users accomplish tasks: eliminate case sensitivity in query fields, add buttons for frequent and rudimentary tasks in spite of NeXTSTEP guidelines to use menus instead, and allow users to select how results are sorted (e.g., by alphabetical order or date as opposed to rank).

The modular construction of MARIAN makes it relatively easy to effect the necessary changes; most

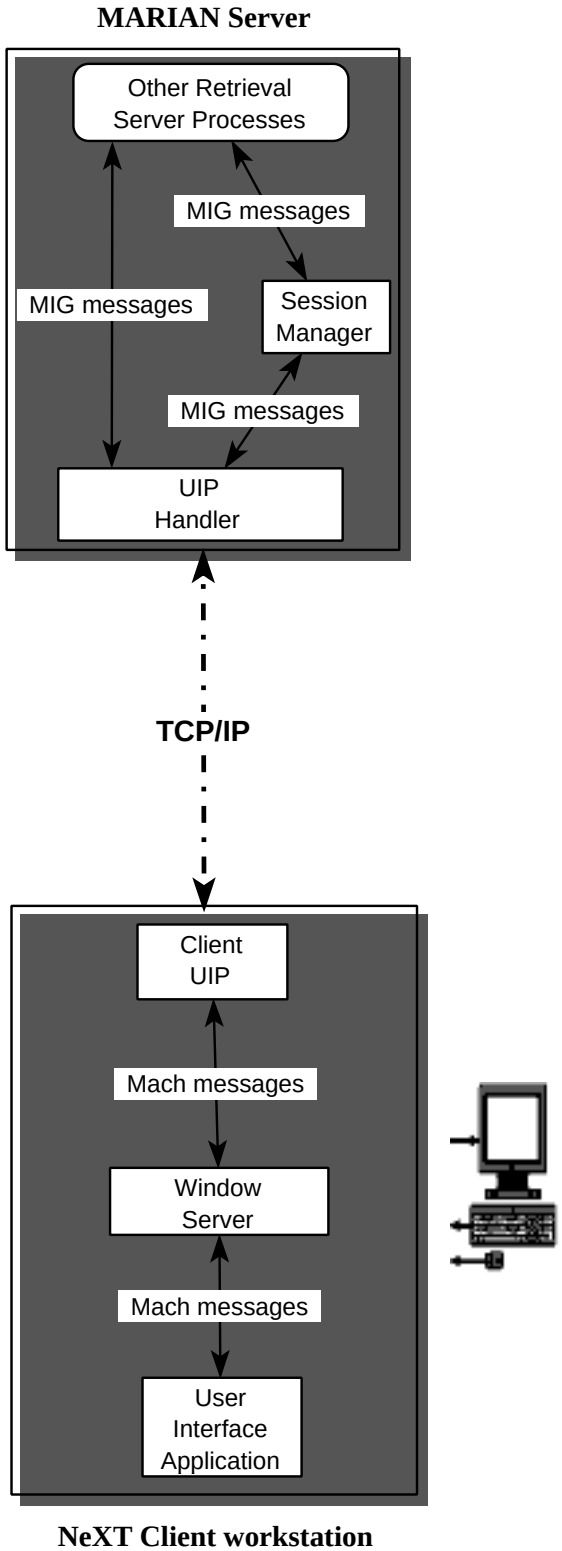

Figure 11: Client/Server Interprocess Communication.

have already been made. Further testing with larger and more diverse groups will lead to iterative refinement of the interface and server.

5.3 Current Status and Plans

The present version of MARIAN matches the design given in Figure 7. Feedback searches, the X/Motif interface, the CURSES interface, and Annotation are the top priorities for development efforts. Regarding data, the testing to-date has involved a subset of the MARC records, numbering roughly 40K. Loading of the full collection of roughly 900K documents should be completed in January 1993, at which time alpha testing will be opened to campus NeXT systems.

As part of completing MARIAN we will extend the current skeletal help facility with context sensitive, point-of-need help and online tutorials [29].

A running version of the MARIAN system will be shown at *SIGIR '93*, as part of the demonstration segment of the program proposed by Professor Philip Smith. By that time, the system should be in beta testing on campus, open to those with NeXTSTEP or X/Motif capable systems. Production deployment, taking advantage of several other interfaces that are under development, is scheduled for Fall 1993, to be open for both Virginia Tech and other Internet users.

6 Future Plans

There are many prospects for MARIAN, including logging of data regarding user interaction for tuning and research, adding other retrieval methods, developing a Z39.50 interface, supporting other databases, and making the system more "intelligent."

One promising line of future investigation builds upon work at Miami University of Ohio. We have provided data from the current 40K record database for experiments in clustering [30]. Results look promising, and we expect to run the fast clustering algorithm involved [6] on the full collection. If successful, this may lead to an additional type of browsing and/or searching.

In parallel with work on MARIAN, LEND is being enhanced further in connection with a Masters thesis that should be completed by Summer 1993, and which will involve experimentation with the MARIAN database [2]. A powerful query language and

efficient optimizing interpreter will be available, supporting operations of hypertext, information retrieval and multimedia applications. These benefits should lead to greater efficiency as well as reduce development costs connected with MARIAN's Database Management Layer.

Finally, MARIAN may be extended to support full-text and hypertext publications including online computer manuals and publications distributed by ACM (in connection with Project Envision, our effort to develop a user-centered hypermedia database from the Computer Science literature [5]).

7 Acknowledgments

Many have helped with the work discussed. At the Virginia Tech Library, Linda Wilson was the main contact helping with REVTOLC, and Charles Litchfield has been our main contact regarding MARIAN. William Dougherty has provided data from the VTLS system for our experimentation.

Tim Rhodes at the Virginia Tech Computing Center did the initial design and implementation of the NeXTSTEP client for MARIAN. Programming and related work on MARIAN has also involved Steve Teske, as well as a number of other staff members at the Computing Center.

Numerous students have assisted in connection with their Masters projects and theses. Ajay Wadhawan and Whay Lee helped with the first REVTOLC studies. Rajesh Ramachander developed a performance monitoring tool for NeXTs that was designed for MARIAN.

Recent usability testing was undertaken by Lucy Nowell, with guidance and assistance from Deborah Hix. Thanks also go to all the individuals who have been subjects in the REVTOLC studies and in our testing of MARIAN.

References

[1] N.J. Belkin et al. Taking account of users tasks, goal, and behavior for the design of online public access catalogs. In *Proceedings of the 53rd ASIS Annual Meeting, ASIS '90*, pages 69–79, Toronto, Nov. 4-8, 1990.

[2] Sangita C. Betrabet. A query language for information graphs. Master's thesis, Virginia Tech, Dept. of Computer Science, to appear in 1993.

[3] Andrew D. Birrell and B. J. Nelson. Implementing remote procedure calls. *ACM Trans. on Computer Systems*, 2(1):39–59, 1984.

[4] Christine L. Borgman. Why are online catalogs hard to use? Lessons learned from information retrieval studies. *Journal of the American Society for Information Science*, 37(6):387–400, November 1986.

[5] Dennis Brueni, Edward A. Fox, Lenwood Heath, Deborah Hix, Lucy Nowell, William Wake, and Bailey Cross. What if there was desktop access to the computer science literature? In *Proc. ACM 1993 Computer Science Conference, CSC '93*, Indianapolis, IN, February 1993.

[6] Fazli Can. Incremental clustering for dynamic information processing. *ACM Transactions on Information Systems*, 11(1):to appear, January 1993.

[7] Qi Fan Chen. *An object-oriented database system for efficient information retrieval applications*. PhD thesis, Virginia Tech Dept. of Computer Science, March 1992.

[8] Ben E. Cline, Robert K. France, and Edward A. Fox. OPAC design document: MARIAN, June 1991. Unpublished Internal Communique.

[9] C.R. Corbin. *The Art of Distributed Applications*. Springer-Verlag, New York, 1991.

[10] Brad J. Cox. *Object Oriented Programming: An Evolutionary Approach*. Addison-Wesley, Reading, MA, 1981.

[11] Amjad M. Daoud. *Efficient Data Structures for Information Storage and Retrieval*. PhD thesis, Virginia Tech Dept. of Computer Science, 1993. To appear.

[12] E. A. Fox. *Extending the Boolean and Vector Space Models of Information Retrieval with P-Norm Queries and Multiple Concept Types*. PhD thesis, Cornell University Dept. of Computer Science, August 1983. Available from University Microfilms Int.

[13] Edward A. Fox. Development of the CODER system: A testbed for artificial intelligence methods in information retrieval. *Information Processing & Management*, 23(4):341–366, 1987.

[14] Edward A. Fox. Testing the applicability of intelligent methods for information retrieval. *Information Services and Use*, 7(4-5):119–138, 1988.

[15] Edward A. Fox. Advanced retrieval methods for online catalogs. In *Annual Review of OCLC Research, July 1989 to June 1990*, pages 32–34. OCLC Online Computer Library Center, Inc., Dublin, OH, 1989-1990.

[16] Robert K. France. User interface objects for CODER, INCARD, and MARIAN, August 1992. Unpublished Internal Communique.

[17] Nasser K. Ghazi. Development of a user interface for the MARIAN system and a server for the SMART system. Master's thesis, Virginia Tech, Dept. of Computer Science, September 1991.

[18] D. Harman, E. A. Fox, R. Baeza-Yates, and W. C. Lee. Inverted files. In W. Frakes and R. Baeza-Yates, editors, *Information Retrieval: Data Structures & Algorithms*, pages 28–43. Prentice-Hall, Engelwood Cliffs, NJ, 1992.

[19] H. Rex Hartson and Deborah Hix. Human-computer interface development: Concepts and systems. *ACM Computing Surveys*, 21(1):5–92, March 1989.

[20] Charles R. Hildreth. Online public access catalogs. *Annual Review of Information Science and Technology*, 20:233–286, 1985.

[21] Charles R. Hildreth. Beyond Boolean: Designing the next generation of online catalogs. *Library Trends*, 35:647–667, Spring 1987.

[22] Charles R. Hildreth. *Intelligent Interfaces and Retrieval Methods*. Library of Congress, Washington, D.C., 1989.

[23] Ray Larson. Classification clustering, probabilistic information retieval, and the online catalog. *The Library Quarterly*, 61:133–173, April 1991.

[24] William H. Mishco and Jounghyoun Lee. End-user searching of bibliographic databases. In Martha E. Williams, editor, *Annual Review of Information Science and Technology*, volume 22, pages 227–264. Elsevier Science Publishers, New York, NY, 1987. ISBN 0-444-70320-0.

[25] N. N. Mitev. Human computer interaction and online catalogues. In *OPACs and Beyond, Proceedings of a Joint Meeting of the British Library, DBMIST, and OCLC*, 1988.

[26] NeXT Computers, Inc. *NeXTstep Concepts*. NeXT Computers, Inc., 1990.

[27] Eskinder Sahle. Development of a user interface for MARIAN and CODER systems. Master's thesis, Virginia Tech, Dept. of Computer Science, January 1993.

[28] G. Salton, E.A. Fox, and H. Wu. Extended Boolean information retrieval. *Communications of the Association for Computing Machinery*, 26(11):1022–1036, November 1983.

[29] B. Shneiderman. *Designing the User Interface: Strategies for Effective Human-Computer Interaction, 2nd ed.* Addison-Wesley, Reading, MA, 1992.

[30] Cory Snavely. Implementation of a cover coefficient-based incremental clustering algorithm for very large document databases. Technical report, SAN Departmental Honors Program, Miami Univ. of Ohio, Dec. 16, 1992.

[31] M. Yee. System design and cataloging meet the user: User interfaces to online public acess catalogs. *Journal of the American Society for Information Science*, 42(2):78–98, 1991.