VNS Retriever: Querying MEDLINE over the Internet

Kevin Brook Long, Jerry Fowler, Stan Barber – Baylor College of Medicine

ABSTRACT

Academic medical centers around the country are developing networking infrastructures and connecting to the Internet. Baylor College of Medicine is developing VNS Retriever, an architecture for comprehensive handling of our institution's database requests. The first implemented instance of this architecture is the MEDLINE Retriever, a tool to query the considerable citation database of medical literature at the National Library of Medicine. Response times that we have experienced with the MEDLINE Retriever have pleased us and impressed our user community. The system will work for small sites, but is extensible for use on large campuses as well. The MEDLINE Retriever uses the Corporation for National Research Initiatives' ABIDE gateway. The principal user interface employs Motif and the X Window System.

Introduction

The information consumer in the academic medical center environment is faced with a modern dilemma; as electronic, networked informatics resources become more common, users are presented with a disparate collection of interfaces, mostly designed independently as islands of information whose only integration with each other is the coincidence of existing on the same network. Although the presence of high-speed networks might appear to bring this electronic information nearer the user, it is instead often unreachable behind a wall of technical detail. We are attempting to develop and apply informatics architectures that more seamlessly integrate the information from certain classes of these resources into the information user's environment. Specifically, we have applied the concept of the layered architecture and the intelligent retrieval engine to facilitate interaction with networked bibliographic resources.

At Baylor College of Medicine, our participation in the National Library of Medicine's Integrated Academic Information Management System (IAIMS) program has involved exploring ways to apply advancements in computing and informatics to the problems of biomedicine. We have developed an integrated networked environment for coordinating human, computer, and informatic resources. The centerpiece of our work is the Virtual Notebook System (VNS), an electronic analogy to the traditional paper notebook [1,2,3]. The VNS is the realization of an architecture that supports the collaborative research environment. The work described here is a natural extension of this effort, a step closer to the longer-range goal of establishing intelligent, networked information clearinghouses.

Recognizing the growing demand in our biomedical research community for rapid access to remote electronic databases, both bibliographic and otherwise, we have devised VNS Retriever, an architecture to facilitate user access to a variety of local and remote electronic databases. The cornerstone of VNS Retriever is a central resource server that acts as an agent between the user and the NLM via asynchronous electronic message passing. At the front-end, we have implemented a user interface written using the X-Window System and the OSF Motif toolkit so that many users of a wide variety of platforms can access this new service [4,5]. As a back-end connection to the MEDLINE database, our resource server makes use of the ABIDE gateway and Knowbot Operating Environment developed by the Corporation for National Research Initiatives (CNRI) [6].

This paper discusses VNS Retriever, a layered architecture for wide-area database query and retrieval, and our first implementation of VNS Retriever, the MEDLINE Retriever. In the next section, we amplify on the motivations for this project. Thereafter, we describe the architecture of VNS Retriever, including the communications interfaces between components. Then we discuss the implementation of the MEDLINE Retriever in order to clarify the structure of the central components of VNS Retriever. We then discuss administration of the MEDLINE Retriever, and present some simple performance analysis, before discussing future work and drawing conclusions.

Motivation for VNS Retriever

The MEDLINE Retriever began as an effort to facilitate Internet access from networked workstations to the National Library of Medicine's MEDLINE facility. It has become the embodiment of an architecture to treat bibliographic information in a

consistent fashion, and to assist the user in overcoming many of the technical details that often impede bibliographic research. At a more general level, VNS Retriever is an architecture that attempts to add (among other things) systemized resource characterization, automated query generation, and intelligent filtering mechanisms to the overall confines of the VNS architecture, which by contrast deal more with the integration, annotation, organization and dissemination of information.

We anticipate that VNS Retriever will be applied to a variety of resource types, including bibliographic (such as is addressed by the MEDLINE Retriever), genetic (such as the GENBANK Retriever), programmatic, textual, video, etc. The MEDLINE Retriever is our first instantiation of this architecture. The advantages that the MEDLINE Retriever brings to MEDLINE access are multiple:

- Database access software can be maintained centrally and singly for all networked users at a given site, allowing updates to search strategies, the MeSH vocabularies, access methods, etc. to be implemented immediately. This greatly simplifies the distribution of software and documentation updates.
- Users need only an X-capable display and a connection to the network to begin using VNS Retriever. It is no longer necessary for a user to procure a modem, a copy of the access software itself (such as Grateful Med [7]), a list of network access numbers, and perhaps even a personal NLM account.
- Users of hundreds of other presently unsupported computing configurations would be able to gain access to MEDLINE.
- This centrally-maintained, globally-available access to the NLM's resources could help to eliminate an institution's need to acquire a local version of MEDLINE.
- Users who travel frequently can maintain access to their queries and citations from anywhere on the network. This is consistent with a model that allows the entire network to become a support mechanism for the user, not just one machine.
- Institutional database usage patterns could be monitored centrally with the goal of understanding and providing better for the needs of the institution's research community and possibly optimizing access methods for the most used resources.
- Frequently in the academic environment, only by enriching the networked environment will users be attracted to a more distributed computing model; we feel that the MEDLINE Retriever adds value to institutionalized networking.

Architecture Of VNS Retriever

The architecture created to accomplish this task is not simply a MEDLINE-specific interface. Logically speaking, the VNS Retriever architecture has 4 layers, as depicted in Figure 1. User interfaces communicate through an application program interface with a user agent called the query manager. The query manager maintains the user's environment, storing queries and presenting their results in the manner desired by the user. The user agent, in turn, submits queries to a server agent called the resource server. The resource server selects the appropriate resources for each query, translates the query appropriately for the resource, and initiates a resource query engine to communicate with the resource. At the bottom layer are the query engines themselves, which are specialized to communicate with the individual resources.

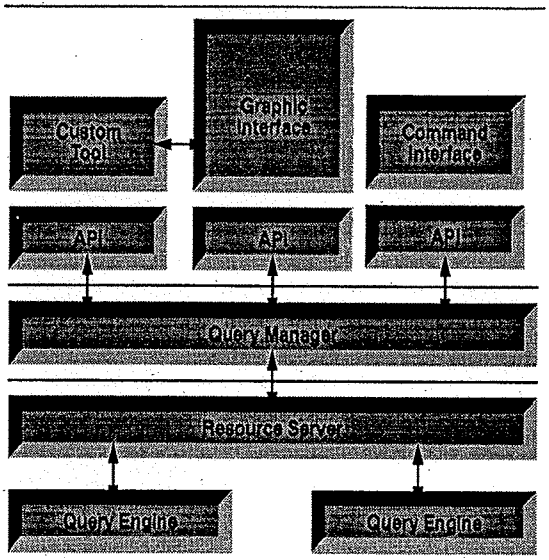

Figure 1: Logical Layout of VNS Retriever

Communication within the architecture necessitates the use of a standard format for queries. We use what we refer to as our canonical query form to define the logical structure of a query and its response. The purpose of the canonical form is to provide a common internal structure for use in mapping terms as displayed by a user interface into terms appropriate to diverse databases. For example, where MEDLINE's Elhill query language uses the indication "(AU)" to identify queries on author names, another local database in common use uses ".au" and the ANSI standard Z39.50 specification uses "ua". On the other hand, since there is no reason to be cryptic in displaying this information to the user, we display the term as "Author" (the substitution of some language other than English for these terms would be straightforward).

The canonical form also considers the type of data to be retrieved. Among the types that we intend to provide in addition to ASCII text are images (of numerous types) and possibly structured data such as those from gene sequence databases. The data recoverable by the MEDLINE Retriever are all textual data, so the only type currently supported is ASCII strings.

The query and presentation terms we use are based on the Z39.50 standard.

A physical view of the VNS Retriever architecture is found in Figure 2. Numerous invocations of the various user interfaces communicate one-to-one with one of several query managers running at the institution. Each query manager serves a specific work group of users related to some administrative entity within the institution; such an entity would typically be a grant-funded research group.

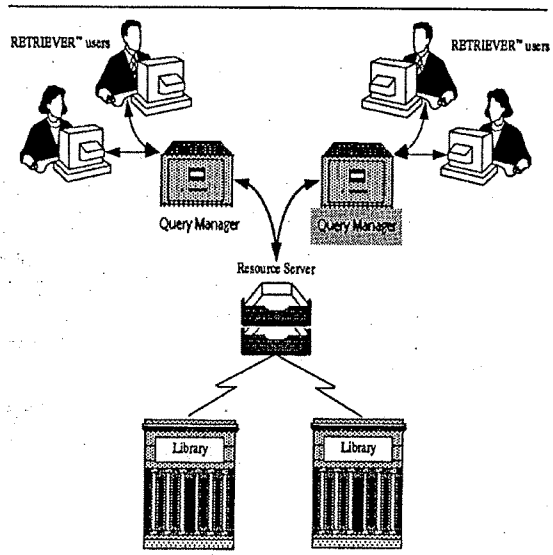

Figure 2: Physical View of the VNS Retriever

Each query manager in turn communicates with the single institution-wide resource server; the resource server is then able to invoke one or more of several resource query engines, each of which may serve more than one database.

The resource server and query manager are the two central pieces of the architecture. We shall describe them first, and then discuss interprocess communication within the system. Since the user interface and resource query engines are implementation dependent, we defer discussion of them to our detailed description of the implementation of the MEDLINE Retriever.

The Resource Server

The server agent is called the resource server. Its administrative role is to centralize institution-wide access control and accounting. This permits us to address a pet peeve in our user community; that is, the need to enter a user id and password for every invocation of a controlled-access program. Access to the resource server is restricted to a set of authorized machines. Access to individual machines is controlled by the machine's protection mechanisms. Accesses from specified machines can be further restricted to a subset of the available resources by means of a resource management table. Our design anticipates that installation on our campus would have a single resource server providing service to the entire organization.

The resource server's role in database retrieval is to provide a common interface to the diverse set of resources that it can support, allowing a single user interface to access all the supported resources. To this end, it supports requests to enumerate the databases it serves and to list the query and presentation terms valid for each. When the resource server receives a search request, it stores the request in its internal database, which is a simple tree with branches for each work group. The server then analyzes the request to determine which database to query. At this point it performs whatever accounting and access control are appropriate to the database, and then translates authorized requests into the appropriate terms for that database. Finally, the query engine for that database is invoked through a set of library routines that launch the query engine in a synchronous exchange, and then retrieve its results when they return asynchronously some time later.

After a query engine returns a response, the resource server notifies the appropriate client query manager. When the client has acquired the result data and acknowledged its receipt, the query and the response are deleted from the local database. Any cost accounting to be done for a given database is performed at this point.

The Query Manager

The user agent is called the query manager. Its administrative role is to provide middle- to long-term storage of queries and results for its work group, thus placing the responsibility for disk utilization on each work group, which runs its own query manager.

The query manager provides the user interface with a set of mappings to translate queries from user display form into canonical form, and to translate results from canonical form to the user-preferred display. It also provides the user interface with the list of available database resources and their query terms. The query manager connects to the resource server to submit queries. The query manager holds responses it receives from the resource server until they are deleted by request of the end user via whatever front end the user may use.

In addition to its intermediary role in database retrieval, the query manager maintains the user profiles for its clientele. These profiles describe what information the user wishes to retrieve as a default, and how it should appear. They also describe translations, as well as the filters and destinations to be used for exporting results to other places.

The query manager maintains little state concerning on-demand search requests. The query manager knows when it has issued a search request to the resource server, and provides a means to delete outstanding requests It depends on the resource server to maintain other information about the state of an on-demand search. The query manager does, however, keep track of scheduled queries in more detail. Selective Dissemination of Information (SDI) is the name that the NLM gives to searches that are stored centrally at the NLM database for incremental execution on a regular basis, for example, monthly when the database is updated. The fact that the query manager permits query scheduling makes the storage of SDI queries at the query manager possible, reducing the burden on the central database and opening the door for regular scheduling of intelligent wide-area database searches when such searches become viable. It is the query manager's job to queue scheduled queries and submit them to the resource server at the appropriate time.

Interprocess Communication

There are several levels of interprocess communication within VNS Retriever. Communication between the upper levels of the system relies on the use of a canonical form for the description of query terms and results. We shall discuss in turn the communications between the query manager and its user interfaces, between the query manager and the resource server, and between the resource server and its query engines.

The Query Manager Interface

The query manager communicates with its user interfaces through an application program interface whose standard queries include user profile handlers, search manipulation, and administration, as well as a search-result callback.

The functions Query-Profile and Modify-Profile allow the user to list and alter default settings for the Z39.50 terms Large-Set-Lower-Bound and Small-Set-Upper-Bound (the maximum number of matching records for which the user deems it is worth returning any data, and the largest number of records to be returned from a successful search, respectively) as well as desired presentation terms to be returned by a search, such as author, title, journal name, etc. This is also the place in which the cost center for charge accounting is specified. To facilitate exporting result data from VNS Retriever to other programs, in particular the VNS, the query manager supports the functions Query-Export and Modify-Export. These functions allow the user to manipulate defaults for data export from VNS Retriever. Groups also carry default settings for profile and export defaults, so that a new user automatically adopts the standards of the group to which it is added.

The function List-Searches returns a list of searches currently maintained for the user by the query manager, and List-Results returns the results for a given search, if they are available. Searches can also be renamed or deleted. A "force" flag in the delete function allows a search to be deleted irrespective of whether it is currently outstanding at the resource server.

The function Submit-Search has three options, submit on demand, submit scheduled, and submit to save. Saving a search simply records the contents of the query at the query manager so that it may be retrieved for later modification and submittal. A search may be scheduled for a one-time search at a later time, or for regular submittal at a restricted set of intervals such as hourly, daily or monthly (although there is no point at the current time in searching daily or more frequently, because updates to MEDLINE are not that frequent, it is our intention to incorporate local databases such as colloquium notices, for which more frequent regular search *would* be appropriate).

Status queries provide information on the current state and availability of results for the named search. Administration requests include the ability to add or delete users or groups, to request that the query manager quit gracefully, and to request changes in the type of information recorded in the log file.

Since the return of results from the resource server to the query manager is an asynchronous event, a user interface can register a result callback so that it can be informed in a timely fashion of the return of a search, without the need to poll the query manager.

The Resource Server Interface

The query manager behaves as a client of the resource server, sending synchronous requests. The resource server recognizes three basic types of request: search, status, and administration.

The resource server's search protocol is based loosely on the Z39.50 standard. The Init and Init-Response Application Protocol Data Units (APDU's) are used to establish a connection between a query manager and the resource server. The Search APDU is used to initiate a search, and a Search-Response APDU is used to acknowledge the successful receipt and launch of the search, using a return code not specified by the Z39.50 standard to indicate this

initial success. Because the initial response to the search APDU is effectively an acknowledgment of the request, the query manager is free to carry on with other tasks instead of blocking synchronously on the search. This deviation from Z39.50 protocol enables the query manager to multiplex on several clients potentially issuing multiple requests per client.

Upon receipt of search results from the query engine, the resource server issues an announcement to the client query manager by means of an asynchronous Search-Response APDU. A fixed-size portion of the result data is piggy-backed with this announcement. This piggy-back packet is sufficient to return the entire response in many cases. Should the response contain more data than will fit in this announcement, an indicator is set to show that more data remain to be retrieved. It is then up to the query manager to request and retrieve the remainder of the data by means of Present and Present-Response APDU's.

There are certain resource server functions that lie outside the Z39.50 protocol. These include administrative functions, including addition and deletion of users and groups (although the identification of a user is not strictly necessary to the resource server's function, it provides a parallel structure to that found in the query manager's internals).

Status queries return information on the state of the server, as well as specific information about groups users, or queries. An additional status query returns the list of resources supported by the resource server. In our initial implementation, this is a trivial response, since the MEDLINE Retriever supports only one database.

The Query Engine Interface

Communication between resource server and query engine is by means of library calls. It is the query engine's responsibility to provide all communication with the database(s) it serves. Each query engine interface requires a set of translations from the canonical form to the query engine's native query language (although the query engine's language may simply be the query language of the database it supports, this need not be the case for a more general query engine, such as the Knowbot query engine), as well as the routines to initialize and launch a query engine instance; to initialize a response retrieval and to drive data retrieval; and to destroy the query engine instance.

The MEDLINE Retriever

The first instantiation of VNS Retriever has been the MEDLINE Retriever. This system has been tested and used by members of the Baylor research community since February, 1992. The MEDLINE Retriever takes advantage of the ABIDE gateway and Knowbot Operating Environment developed by David Ely of CNRI [8]. The Knowbot query engine is used to provide access to MEDLARS, the host for the National Library of Medicine's on-line bibliographic database. We shall first briefly describe the user interfaces for the system. Then we shall discuss the internals of the resource server and query manager and briefly describe the behavior of the Knowbot query engine.

User Interface

The user interface for the MEDLINE retriever does not reflect all of the generality described for the VNS Retriever architecture, but it provides a basis from which to understand the system, as well as a fully functional MEDLINE query tool that is easily integrable into other environments such as the VNS.

At the top layer we have demonstrated several user interfaces. There is a command-line interface that reads queries from files created by the user, and then formats the results to standard output. Another interface uses a simple form-driven X-Windows front end to the command-line interface, and formats the results into another window that permits simple user browsing through the returned citations.

The interface to which we have paid the most attention is a menu-driven X-Windows interface written using the Motif toolkit. This front end facilitates more sophisticated handling of queries and results, including creating queries using MeSH, querying for the status of outstanding searches and exporting selected citations to the Virtual Notebook System, as well as to order photocopies of desired citations direct from the Houston Academy of Medicine-Texas Medical Center Library. This interface is described in more detail elsewhere [9].

Server Structure

Both the resource server and the query manager maintain a mirror of their internal trees on stable storage. This allows both servers to maintain their state across invocations by reading their respective storage trees found on disk to build their internal trees at start-up. When the resource server initializes, it examines this tree for searches that have not yet received a reply from their query engines; it resubmits such searches, since the return address for the query engine instance associated with the old request is now invalid. When the query manager initializes, it builds a queue of scheduled and regular searches that it finds on disk, and sets a time-out for the shortest interval found among those queued.

We have chosen to store the data on disk in ASCII files, although we could substantially reduce storage by use of binary data. This permits visual inspection of the storage tree to help detect system problems, and simplifies debugging. Duplicating the information on disk requires frequent file writes that certainly have some impact on performance. We could improve this performance in at least one respect: Our current implementation employs a

UNIX file system, creating a directory for each group, user, and query. Replacing this mechanism with a common database manager such as *ndbm* would probably significantly improve our performance, at the expense of losing the ability to inspect the storage tree visually. This is a problem that we will address again when scaling becomes a greater issue than it is now.

Communication Protocols

Figure 3 shows the interprocess communication paths within the MEDLINE Retriever. Each level of the system employs a conventional client-server model for most transactions. At each level, an upcall or callback is also necessary to provide asynchronous notification. The query manager uses SunRPC [10]. Below that, TCP/IP network sockets are used.

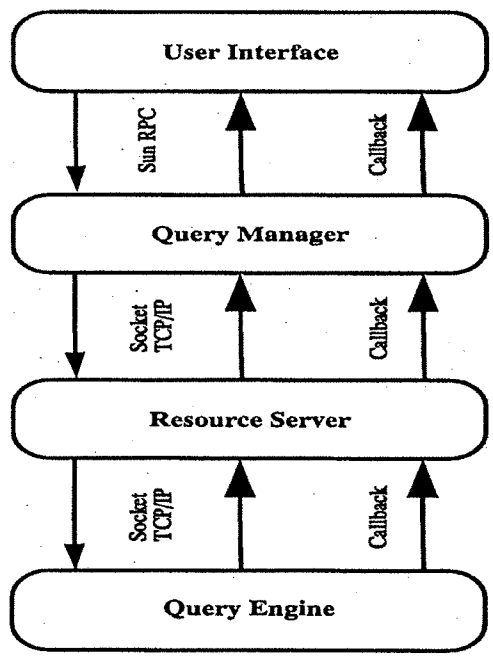

Figure 3: MEDLINE Retriever Interprocess Communication

The Resource Server

The connection between query manager and resource server employs BSD network sockets, using TCP/IP. The resource server opens listening sockets on one front end port and one back end port. The driving loop maintains one global socket, as well as one socket per active client, one initiator socket per query engine, plus one socket per active query.

When a query manager initializes a connection to the resource server, the resource server sets a short time-out after replying. When the time-out expires, the resource server examines its internal tree for outstanding responses belonging to the newly connected query manager. Any such responses have never before been successfully reported to the query manager, so at this point they are dispatched to the query manager with the asynchronous SendResponseAPDU. This greatly simplifies connection initialization.

The Query Manager

The query manager's internal structure has much in common with the resource server. In fact, their executable code links a common library for manipulating the internal storage tree.

Connections between the query manager and the user interfaces utilize SunRPC for interprocess communication. To some extent the use of two different implementations for our communications protocols represents an experiment. We chose to use SunRPC in order to reduce the overhead caused by the use of TCP/IP, but let SunRPC simplify the handling of UDP connections. As a serendipitous effect, this permits us transport independent communication.

Although other RPC mechanisms are available, we chose SunRPC without examining other alternatives simply because it was freely available on our initial target platforms. We have found SunRPC to be congenial, although development was more difficult than it might have been because there is no checking for null string pointers in the external data representation library.

Because the query manager is effectively in the middle of a pipe, acting as a server on one side and as a client on the other, it cannot use the SunRPC svc_run() call to drive its loop. The query manager opens its front end by means of SunRPC initialization, and also opens a back end socket with which to connect to the resource server. The driving loop simply deals with the SunRPC communications first, thereafter responding to any asynchronous responses from the resource server.

Because of the differences in treatment of SunRPC between the procedural model of a "normal" UNIX program and the event-driven model of an X Windows application, there are two slightly different application program interfaces available that handle the two cases.

To handle regularly scheduled SDI searches, the query manager builds a queue, and sets an alarm for the shortest interval found among the scheduled searches. The action taken at alarm time is simply to set a flag to indicate the elapse of the interval. When the query manager falls to the bottom of its driving loop, it tests this flag and invokes the queue handler if set. This avoids the possibility of inconsistency in the internal tree, because there is no access to internal data structures during the execution of the asynchronous alarm handler.

Help Server

Additionally, the system provides a help service. On-line help is available for most objects in the graphic user interface. In order to provide maximum flexibility and host independence, this help is provided by remote procedure calls to a help service. Help requests are keyed by program and function name. The contents of the file corresponding to the key are returned to the requester as an ASCII text object. The requesting interface can display the information as it sees fit (button-specific helpfiles probably make sense for only one interface, of course). This service is isolated from the query manager in order to permit the support of multiple related programs out of the same help service.

Query Engines

Although we anticipate several distinct query engines, we currently support only the Knowbot Operating Environment.

The Knowbot Operating Environment

In the specific case of the MEDLINE Retriever, the query engine is provided by the pioneering work done by David Ely at CNRI in creating the ABIDE gateway to MEDLARS and the Knowbot Operating Environment. This gateway receives appropriately constructed Knowbot queries, validates and executes them and return the results to the Knowbot constructor that sent them.

Our collaboration with CNRI on the VNS Retriever project has provided CNRI with an opportunity to shake down the concepts involved in the Knowbot Operating Environment. At the same time, it has allowed us to work directly on creating an environment for campus use that could be replicated in other campus environments where access to centralized databases is important.

The ABIDE gateway was originally created by CNRI as a demonstration project for the NLM. Its user interface closely duplicated the functionality of the MS/DOS version of Grateful Med, but provided no query management features or asynchronous access. Through this cooperative effort, we were able to make use of the back end functionality while enhancing the power of the user interface.

The ABIDE gateway runs on a Sun-4 machine located at the NLM. It serves to connect the IBM environment on which the MEDLINE database is operated to the Internet.

The transactions between the resource server (via the Knowbot library) and the ABIDE Gateway use TCP/IP. As described above, the Knowbot query engine is launched in one synchronous action. The launch of the Knowbot query includes a return address to which the ABIDE gateway will subsequently respond asynchronously. Each query from a given MEDLINE account is queued at the ABIDE gateway and performed sequentially.

Our use of the Knowbot Operating Environment does not exploit its ability to return data to a receiver on a different host from the launcher. This design decision is based on our desire to control access and perform accounting centrally.

Other Query Engines

In the case of our first implementation, which involves internet communication, initialization and launch comprise the first step of the search process, and at this point the resource server reports a successful launch to its client. Thereafter, the server responds to activity on the retrieval socket by invoking the query engine's data-retrieval function. Anticipated future query engines that involve local area access may have essentially null query retrieval functions, since it is believed that data response will be quick enough to incorporate in the launch phase. Such queries will also bypass the stage involving recording on stable storage for the sake of speed.

Among the query engines that we are considering for addition are a Wide Area Information Service (WAIS) engine, and an RPC-based Unified Medical Language System (UMLS) engine [11].

Administration

Administration of this system occurs at two levels. The Query Manager and the Resource Server each have separate administrative requirements and can be administrated separately as needed. The architecture provides for one Resource Server per site and one or more Query Managers.

Resource Server Administration

Administration of the Resource Server involves checking accounting information for each request submitted. In order to give system managers at different sites local control over how the resource server does accounting, we provide a hook for a remote procedure call to an external server. This server simply returns zero for access accepted and non-zero for access rejected. The server does its permission checking by reading a script written by the system manager to suit the needs of the individual institution.

Additionally, the Resource Server verifies information supplied by the Query Manager about the group or the user submitting the query to insure that the group or user is authorized to make use of the resource to which the query is directed. If the user or group is authorized, the query is processed. If not, the Resource Server returns an **ACCESS DENIED** reply in response to the query. Password management and all other resource access information is stored in the Resource Server access database.

Query Manager Administration

Management of the Query Manager centers on the disposition and storage of queries and query responses. Most of the management of the Query

Manager occurs with each user authorized to make use of it. Individual users enter queries and determine how to store responses.

However, there are certain matters that require an administrator. Authorizing users to make use of a particular query manager, removing old users (including purging any queries or query responses created by those users) and providing a default user profile must be done by the Query Manager Administrator.

Performance and Evaluation

The performance of the MEDLINE Retriever has impressed our user group. Response times are good from half a continent away. Although the system has many good qualities, there is ample room for improvement.

System Performance

In a battery of timing tests we ran using the command-line user interface, we measured both the time required by the resource server, and the round trip time from command-line user interface to MEDLARS and back. At the resource server, the average response time (from initialization of query engine instance to completion of data retrieval) was 11.7 seconds for a search that returned 10 citations (author, title, and source only) from a query involving text search of two words. The average response time was 18.5 seconds for the same search to return 200 citations, which required transmission of about 70 Kilobytes of data (we impose a limit of 200 on Small-Set-Upper-Bound, the maximum number of records to return, which is a compromise between the belief that we share with some librarians that one generally does not wish to examine any results at all until one has pared the result set down to a large handful, and the use-patterns and desires of many of our scientific users, who do not mind browsing even hundreds of titles in search of relevant citations).

The round-trip times at the command line interface were less satisfying. The average response time in the case of returning 10 citations was 15.9 seconds, an average four second overhead for the system architecture. Returning 200 citations took an average of 27.4 seconds, an average 9 second overhead. Naturally, virtually all of this overhead is clock time, not CPU; unfortunately that is what the user perceives. Command line response time is affected by the times incurred by both query manager and resource server in polling their sockets during select() calls, as well as time required to update status files. We intend to eliminate the causes of this excessive overhead in the near future.

During normal use, the average response time for a successful search was 16 seconds. Most searches (292 of 324) required less than 30 seconds to return. Of those, the average was 9.6 seconds. The longest successful search required 5 minutes 23 seconds.

The average time required to initialize a query engine instance was 1.2 seconds, which involves opening a socket on the local host and establishing a connection to the ABIDE gateway. The maximum initialization time was 9.9 seconds.

The average cost of these searches as reported by the ABIDE gateway was $2.30. A simple search that returns little or no data costs around a dime, but any attempt to retrieve larger amounts of data, either to retrieve abstracts, at around 3 Kilobytes apiece, or to retrieve numerous citations, causes the charge to increase rapidly. This has raised the issue of how to account for MEDLINE searches. The cost schedule that is currently used by the NLM is perceived as prohibitive by our users.

Evaluation of the System

Overall, we are pleased with the results of our design. One question that remains to be resolved is whether a single resource server will scale well for full-scale use at a large institution. Baylor has about forty specialized research centers comprising around eight hundred laboratories with several thousand scientists participating in research, and another several hundred involved primarily in education. A more detailed analysis of response time under a heavy load is needed to determine whether a single resource server can in fact support the whole community at Baylor.

Another potential problem with respect to MEDLINE specifically is that queries issued on a single account are queued. Since our current practice is to issue all queries on one account, this can adversely affect users' response times. Although our actual deployment policy is as yet undetermined, we may choose to employ multiple MEDLINE accounts to circumvent this situation if it actually causes difficulty.

With regard to our decision to logically isolate our resource server from the query engines, there are advantages and disadvantages. On the one hand, much of the knowledge necessary to access diverse databases exists already in the Knowbot Operating Environment, and our logical separation from that environment represents some duplication of effort. On the other hand, use of our own canonical query form permits us to translate to and employ other query engines with minimal effort; it also permits us to consider the use of lighter weight mechanisms for inherently small local databases, such as "bcm.seminars", the college's electronic colloquium bulletin board. Overall, this logical isolation provides a flexibility that is a greater boon than a hindrance.

We chose to use our own canonical form in preference to SQL partly from convenience, and because many resources which we wish to access are not in a relational form for which SQL is well-suited. Similarly, we do not simply employ a free-text-searching algorithm in order to take advantage of the structured nature of many common resources.

As for our implementation, there are several things we will likely change in a future release of the system. First, we would like to replace the existing socket-based transport used in the resource server with an RPC model. This would take advantage of the RPC-based communication protocol that already exists in the query manager. Most of the fundamental requests in both servers are identical in structure, if not in effect, so the parallelism of design would be reflected in reuse of code (the original specification of the query manager used socket-based communications for that reason).

We had chosen the socket model for the resource server's communications in part to take advantage of the Z39.50 library available from Thinking Machines Corporation's WAIS project [12]. Since the Z39.50 protocol does not handle the problem of asynchrony we have mentioned earlier, there is some disadvantage to adhering to that model. There is, however, substantial benefit to the natural way in which the SunRPC package wraps around our canonical forms.

Finally, it might behoove us to take responsibilities away from the user interface. Many of the functions that it performs would be better performed by small utility routines that it might invoke in the course of its execution.

We are pleased with the use of SunRPC, although more gracious handling of null string pointers in the external data representation code would be welcome.

Although we considered the use of the Sun lightweight process library, the benefit that it would have provided us in terms of imposing a logically multi-threaded design was rendered null by the problems to be overcome in compensating for the fact that a single lightweight thread blocking on I/O blocks the entire process. We also had difficulties with incompatibility in the memory allocation.

Related Work

The NLM has recently promulgated an Internet-capable version of Grateful Med. This is an excellent system for its purpose, which is to provide user-friendly support for MEDLINE access; however, it exists of itself, and does not integrate well with the goals of IAIMS at Baylor.

We also wish to reiterate our respect for the work of CNRI, whose Knowbot Operating Environment is the foundation of our link to MEDLARS at NLM. Their coop- eration has contributed markedly to our success. The Knowbot query language provides a single Z39.50-like query interface to each of the databases supported by ABIDE. It is an ideal back end for our work, because there is great potential for the expansion of its powers in order to expand ours.

MCC's Carnot project is a vastly more ambitious system that incorporates both database retrieval and update, employing a transaction shell called Rosette that uses the Actor model for organizing asynchronous distributed database accesses [13]. The development of a Carnot query engine could significantly extend the power of our system.

The WAIS project takes a different view of wide area data retrieval, employing full-text search instead of keyword indexing. This is a technology ideally suited to the parallelism of the Connection Machine for which it was originally designed. The user interface to a WAIS service could be subsumed by our existing user interface.

The University of Minnesota's **internet gopher** project implements a document delivery service. It allows users access to various types of information residing on multiple computers in a seamless fashion. The interfaces for gopher use a hierarchy of menus. Each menu item may provide access to other menus, files, or gateways to other types of information servers includes the WAIS environment described above.

Future Work

The design of MEDLINE Retriever has laid the foundation for several further development efforts. These include the augmentation of the powers of the back end, closer integration with the VNS, and expanding support for alternative data types.

Back End Support

We intend to expand both the number of query engines available to the resource server, as well as to include support for several other resources. Among the resources we hope to add are Current Contents, as well as a service to index colloquium announcements and other institutional events. Standing queries for these latter resources can be established to report upcoming events of interest to each user. We hope to deploy a local ABIDE gateway in order to experiment with the use of Knowbot scripts for access to these resources.

Other resources that we wish to add suggest the construction of new query engines. One of these is a UMLS server to enhance the power of the user interface to the MEDLINE Retriever. Another would be a WAIS query engine.

Closer Integration with VNS

With the ongoing development of version 3.0 of the VNS, we have the opportunity to incorporate wide area hypermedia into the VNS. We have dubbed this new model VNS-STAR. The VNS-

STAR model distinguishes between the page, VNS's traditional vehicle for data organization, and the STAR-PAGE. Objects on a STAR-PAGE contain not data, but the reference information necessary to recover a specific dataset from anywhere on the internet, thus trading bandwidth for storage, and increasing the timeliness of data displayed through the VNS.

Another area of interest is adapting VNS Retriever for the retrieval and display of alternative data types. This is closely tied to the object orientation in VNS 3.0. VNS Retriever will initially recognize the types defined by the VNS, particularly images. Any further expansion of VNS Retriever's type set can be done through the user-definable objects of the VNS. Because of VNS Retriever's use of high-speed networking, it is well suited to take advantage of imagery and other higher-volume data that will inevitably become part of the typical bibliographic resource.

Conclusion

The VNS Retriever has served a useful role in helping us develop an architecture to systemize access to bibliographic and other resources. It is a necessary step towards developing more intelligent networked data retrieval tools. As a tool in its own right, it is performing satisfactorily for the BCM user community, while at the same time, it is serving to illuminate the challenges that lie ahead in our development of more useful tools for the typical researcher.

VNS Retriever is representative of many of our efforts. Even as we attempt to create an overall approach to addressing classes of problems, our user community benefits from the application of a useful tool which addresses a specific need. The VNS Retriever is paving the way for more facile access to the ever-expanding collection of internetworked resources.

Acknowledgments

Support for this work was provided by The National Library of Medicine (N01-LM-1-3516 and USPH-02-G08-LM04905).

Thanks are due to Ed Sequeira of the NLM, and to David Ely and Vint Cerf of CNRI for making our use of ABIDE possible. In addition, we wish to thank Tony Gorry and the rest of the Baylor IAIMS group for their ideas and encouragement. Thanks to Cindy Petermann and Barry Meyer of the group for their helpful comments on the manuscript.

For more information on the Knowbot Operating Environment, contact CNRI electronically at: knowbot_info@nri.reston.va.us

References

[1] A. M. Burger, et. al. "The Virtual Notebook System," *Proceedings of the Hypertext '91 Conference,* ACM, 1991.

[2] G. A. Gorry, et al., "Computer Support for Biomedical Work Groups," *Proceedings of the Conference on Computer Supported Cooperative Work,* September 1988.

[3] G. A. Gorry, et al., "The Virtual Notebook System: An Architecture for Collaboration," *Journal of Organizational Computing,* Volume 1 Number 3, 1992.

[4] R. Scheifler and J. Gettys, *The X Window System,* Digital Press, 1988.

[5] Open Software Foundation, *OSF/Motif Style Guide,* Prentice-Hall, 1990.

[6] R. E. Kahn and V. Cerf, *An Open Architecture for a Digital Library System and a Plan for its Development, The Digital Library Project, Volume 1: The World of Knowbots,* Corporation for National Reserach Initiatives, Reston, Virgina, 1988.

[7] B. Shearer, L. McCann, L. and W. J. Crump, "Grateful Med: Getting Started," *Journal of the American Board of Family Practice,* Volume 3, Number 1, pp. 35-38, January-March, 1990.

[8] David Ely, *An Overview of the ABIDE Gateway System,* Technical Report TR-91-1, Corporation for National Research Initiatives, Reston, Virginia, 1991.

[9] J. Fowler, K. B. Long, and S. Barber "The MEDLINE Retriever," Submitted, *16th Annual Symposium on Computer Applications in Medical Care,* Baltimore, Maryland, November 1992.

[10] Sun MicroSystems, Inc., *RPC: Remote Procedure Call Protocol Specificaion Version 2; RFC 1057,*Network Center (NIC) at SRI International, Menlo Park, California, June 1988.

[11] S. Barber, J. Fowler, K. B. Long, R. Dargahi, B. Meyer, "Integrating UMLS into VNS Retriever," Submitted, *16th Annual Symposium on Computer Applications in Medical Care,* Baltimore, Maryland, November 1992.

[12] Brewster Kahle, *Wide Area Information Concepts,* Thinking Machines Corporation, Technical Memo DR-89-1.

[13] Microelectronics and Computer Technology Corporation, *MCC Carnot Project Description,* Austin, Texas, 1991.

Author Information

Kevin Long is Director of IAIMS Development at Baylor College of Medicine. A graduate of Rice University, Kevin has been involved in the Integrated Academic Information Management System initiative at Baylor since 1985, and is Vice President of The ForeFront Group, Inc., a

technology-transfer company which distributes the VNS outside of BCM. Reach him via U.S. Mail at Baylor College of Medicine; One Baylor Plaza; VPIT; Houston, TX 77030-3498. Reach him electronically at klong@bcm.tmc.edu.

Jerry Fowler has been a member of the technical staff of Integrated Academic Information Management System Development Group at Baylor since 1991. He holds bachelor's degrees in math and music from the University of Oklahoma and a master of science in computer science from Rice University. His U.S. Mail address is IAIMS Development, Baylor College of Medicine; One Baylor Plaza; Houston, TX 77030-3498. His electronic home is gfowler@bcm.tmc.edu.

Stan Barber is Director of Networking and Systems Support at Baylor College of Medicine. A graduate of Rice University, Stan has been involved in the Integrated Academic Information Management System initiative at Baylor since 1986. Stan also supports ongoing development of **rn**, the popular news reader program originally developed by Larry Wall. Reach him via U.S. Mail at Baylor College of Medicine; One Baylor Plaza; Houston, TX 77030-3498. Reach him electronically at sob@bcm.tmc.edu.