Integration of The Organization Engine and Library 2000

Ron Weiss

Brandeis University

May 13, 1992

Abstract

In the contemporary research environment, users access and manipulate information gathered from diverse data sources. The Organization Engine is a prototype being developed at the Cambridge Research Lab of Digital Equipment Corporation for the incorporation of data from disparate sources into a local homogeneous framework. It relies on information management based on the notion of retrieval and manipulation through the organization of the data in a non strict hierarchical structure.

In order to fully exploit the benefits of such an environment, the Organization Engine must provide access to remotely located, search based, information retrieval systems. "Library 2000" is an implementation of such a system aimed at tackling the issues of the fully integrated and distributed electronic library of tomorrow.

This thesis incorporates the remotely located, search based, information retrieval capabilities of "Library 2000" into the data management scheme of the Organization Engine. It utilizes the concepts of unique record identifiers and query links. Two access facilities based on these different notions are implemented from the Organization Engine "client" application to the "Library 2000" "server" application.

Contents

List of Figures

Chapter 1

Introduction

In the domain of information storage and retrieval, computers have proved to be an extremely efficient tool. The majority of innovation in this field has been in the improvement of search based retrieval of pertinent data. There has been a focus on the development of data models that facilitate fast searches on and manipulation of the data objects.

Another approach to information management involves the organization of data in a hierarchical framework. Retrieval of the pertinent information is accomplished by utilizing the user's intimate knowledge of the data infrastructure. The organization of pertinent data according to the users' needs introduces many benefits to the efficiency, expediency, and "usefulness" of information retrieval. Unfortunately, research on an organization-based approach to the management of information has been neglected.

Naturally, a database management system that allows the combination of both methods of information storage and retrieval is desired. A software system that merges organization-based capabilities with a search based engine is proven to be beneficial. It will provide users with the ability to organize, categorize and manage data to their own likings, while also allowing for meaningful searches on large, apparently unorganized, sets of data.

The Organization Engine, the front end of the implementation of this software system, is a prototype currently being developed at the Cambridge Research Lab of Digital Equipment Corporation by Miller and Weiss. It is a data object management and presentation system that provides a common interface to remote and heterogeneous data sources through translation drivers.

The remote data source utilized is a search based information retrieval system. This system is the "Library 2000" project of Professor Saltzer, currently developed at the Laboratory for Computer Science at MIT. This system investigates a multitude of issues relevant to the implementation of library systems of the future. These issues include scalability, presentation client-search engine protocol, value-added indexing, images, reliability, and aspects of a distributed system.

This thesis describes the design and implementation of a prototype integration of the two systems. Two innovative translation services are added to The Organization Engine in order to fuse it with the search engine. Therefore, the Organization Engine is enhanced to also function as a client of the search system. In order to provide the necessary background, this thesis furnishes a brief overview of the two information management systems.

1.1 Goals of Thesis

This thesis investigates several issues that are consequential in the design of a distributed information retrieval system. One of the important challenges currently faced by designers of these applications is a seamless integration of distributed systems, including data repositories and presentation level applications.

The following issues are diagnosed in this thesis:

- What are the possible methods of integration? This thesis identifies and examines five alternatives:

  1. Immediate Duplication
  2. Unique Record Identifiers (in depth)
  3. Query Links (in depth)
  4. Partial Storage
  5. Browsing / Traversal of the search indices

- Issues in distributed database management:

  1. Remote system syntax transparency
  2. The translation of data
  3. Managing the physical distribution of information sources.
  4. Establishing a coherent contract between the two systems
  5. The time of search versus the moment of information retrieval

- Survey related research.

1.2 Overview of Thesis

This thesis describes the integration of the Organization Engine approach to data management with the search based information retrieval system.

In chapter 2, we present some related work in the field of information retrieval.

In chapter 3, we present an overview of the two systems being integrated.

In chapter 4, we present an overview of the rationale and design issues of the unique record identifier service.

In chapter 5, we present an overview of the rationale and design issues of the query link service.

In chapter 6, we present alternative approaches to the integration of the two systems that have not yet been implemented.

Finally, in chapter 7, we present our conclusions regarding the design and implementation of the integration and suggest future research paths.

Chapter 2

Related Research

This chapter provides an overview of relevant contemporary research. The characteristics of such research include the integration of search based and organization-based information retrieval mechanisms, such as the Semantic File System discussed in the following section. Relevant research may also include work related to distributed information retrieval. The prominent research that investigates such issues is the Z39.50 protocol definition for information retrieval services discussed in section 2.2, the Wide Area Information Servers discussed in section 2.3, and finally the World Wide Web implementation that is discussed in section 2.4.

2.1 Semantic File System

The Semantic File System (SFS) project of Gifford at MIT is a contemporary implementation that incorporates search facilities into an organization-based storage system [SFS]. It overlays a UNIX NFS file system with a search and indexing server to provide associative access to the system's contents. The system automatically extracts and indexes attributes from files with type specific transducers. The Semantic File System then enables the user to locate data entities, i.e. files, by formulating queries that describe the desired attributes of the entities.

2.1.1 Virtual Directories

The associative access into the tree structured file system is accomplished through *virtual directories*. The Semantic File System interprets the names of virtual directories as query specifications. Upon request, the system dynamically constructs virtual directories that contain the result of the query. This result corresponds to the set of files and/or directories that satisfy the search criteria.

2.1.2 The Maintenance of Query Results

The Semantic File System allows the user to save a computed set of queries using file save programs such as *tar*. Therefore, it is possible to utilize virtual directories in order to preserve query results. These results reflect the behavior of the search system with respect to a certain time in the past. This approach is similar to the query result maintenance of unique record identifiers (Section 4.1.7).

However, unlike the maintenance of record identifiers, the semantic file system cannot guarantee future access to the data entities. The storage of symbolic links does not insure future access because files can be transferred to other locations in the directory structure. A different approach in the Semantic File System allows the storage of file copies. This archival of file copies stores the information as available in the system at a certain time, rather than providing a true link to the dynamically changing data object. This replication technique is similar to the immediate duplication method discussed in section 6.1.

It is also possible to maintain the query specifications, rather than the query results. "When a symbolic link names a virtual directory, the link describes a computed view of a file system". [SFS, p.19] The notion of Query Links in this thesis bears a resemblance. Chapter 5 elaborates on the maintenance of query links in the Organization Engine. Significantly, it discusses novel issues that arise from the different, highly distributed and heterogeneous environment.

2.1.3 "Heterogeneous Storage (Indexing)", "Homogeneous Access"

The Semantic File System is concerned with the storage, management and retrieval of files. During the storage update phase, the system indexes the data objects to allow future retrieval based upon attribute matching. In this phase, the entities are treated based upon their type (i.e., text document, mail message, object file). The system utilizes file type specific transducers to extract the meaningful indexing information from the files. This approach is therefore regarded as a "Heterogeneous Storage" scheme.

In contrast, during the retrieval phase, all data entities are treated equally since the underlying framework is a file system. The access method is therefore homogeneous in the sense that all files are retrieved with the same apparatus. This approach contrasts sharply with the objective of the Organization Engine, which is to access legacy repositories, or heterogeneous data sources.

2.1.4 Type of Information

Finally, an obvious distinction between the Semantic File System and the system described in this thesis is the nature of the administered information. While the focus of the SFS is a tree structured file system, this thesis is concerned with the management of heterogeneous information gathered from disparate data sources. Unlike the Organization Engine, the SFS does not govern the transformation of data from legacy repositories to a homogenous representation.

2.2 Z39.50

The Z39.50 is an American National Standard for Information Retrieval Service definition and protocol specification for library applications. The protocol allows an application on one computer to query the database of another computer. It specifies the procedures and structures utilized for intersystem communication. The protocol targets "connection oriented, program-to-program communication utilizing telecommunication... It is assumed that the [client] initiates requests on behalf of the a user who wishes to to search a database located on a remote system." [Z39.50] The client computer capabilities include the submission of search requests, petitions for database record contents retrieval and responses to miscellaneous control requests by the data server.

2.2.1 A Session – Maintaining State

The Z39.50 protocol defines three phases during the life of the applications' interconnectivity: establishment, information transfer, and termination. Moreover, during an established session, the server computer stores substantial state information regarding the status of the client application. The client relies on the maintenance of this state by the server for the retrieval of pertinent information and for the query refinement process. The client may not duplicate or store this state information to utilize it in the future retrieval of records. In contrast, this thesis postulates that an intersystem communication protocol relying on the maintenance of state fails to achieve sufficient robustness (see section 3.2.2).

2.2.2 Result Sets versus Unique Record Identifiers

The protocol utilizes the notion of a result set maintained by the server to enable the client to specify records for retrieval. The client submits a search to the database server. The server then establishes an ordered set composed of the records selected by the search request. The logical structure of this result is a named, ordered list. To retrieve a record's content, the client specifies a sequential position within the set. This method of interaction lacks the notion of permanent and unique record identifiers. This thesis argues that the server should present record identifiers to the client application. The client may then utilize these identifiers in order to specify which records to retrieve. Section 4 discusses the importance of unique record identifiers within the context of intersystem information retrieval.

2.3 Wide Area Information Servers

The Wide Area Information Servers (WAIS) project headed by Brewster Kahle of Thinking Machines, Inc., addresses the problem of finding, selecting and presenting information from remotely distributed data sources. It establishes an application level protocol for query and retrieval of information. The protocol employed is a variant on the Z39.50-1988 protocol discussed previously. However, it supports only the initialization and search facilities of the standard. The WAIS protocol disregards the other five facilities since they are no longer essential.

The WAIS project implements concepts similar to the ones realized in this thesis. It utilizes the notion of unique record identifiers to access and retrieve remote records. The WAIS implementation also does not require the server to maintain any state for the client application. Finally, similar to query links, WAIS dynamic folders embody a query specification and store a set of record references.

However, the WAIS project does not fully exploit the unique record identifiers and query facilities. Providing such capabilities in an organization-based framework is more beneficial than in the flat structure of the WAIS user interface. For instance, one cannot store *hypertext links* (unique record identifiers) to pertinent data objects. Section 4.1 elaborates on the merits of being able to store links to remote and dynamically changing records. It is also advantageous to be able to freely and flexibly copy and/or move references to remote records within an organization framework. For instance, the user can group such references into personally meaningful categories, or store them in desirable and easy to locate destinations for future access. In addition, the WAIS documentation does not fully discuss the significance of unique record identifiers. [KAHL90]

Finally, the WAIS metaphor currently allows the client to access only WAIS servers. Therefore, the premise is that the data sources will conform to the WAIS client protocol specifications. This thesis takes a different approach to information retrieval from disparate data sources. Because of the existence of numerous data sources, the client conforms to the server requirements by providing a translation service for each data source. The translation to a homogeneous representation is done locally, rather than changing the data source format. This approach is much more flexible than the WAIS process, as it allows database developers to freely modify and enhance the data sources. When new innovative data sources appear, a new translation driver can be easily rewritten, enhanced, or modified. Thus, our implementation also deals with one homogenous representation, but is capable of combining information from disparate data sources. (In fact, a translation driver that interfaces to a WAIS system can be easily added to the Organization Engine system).

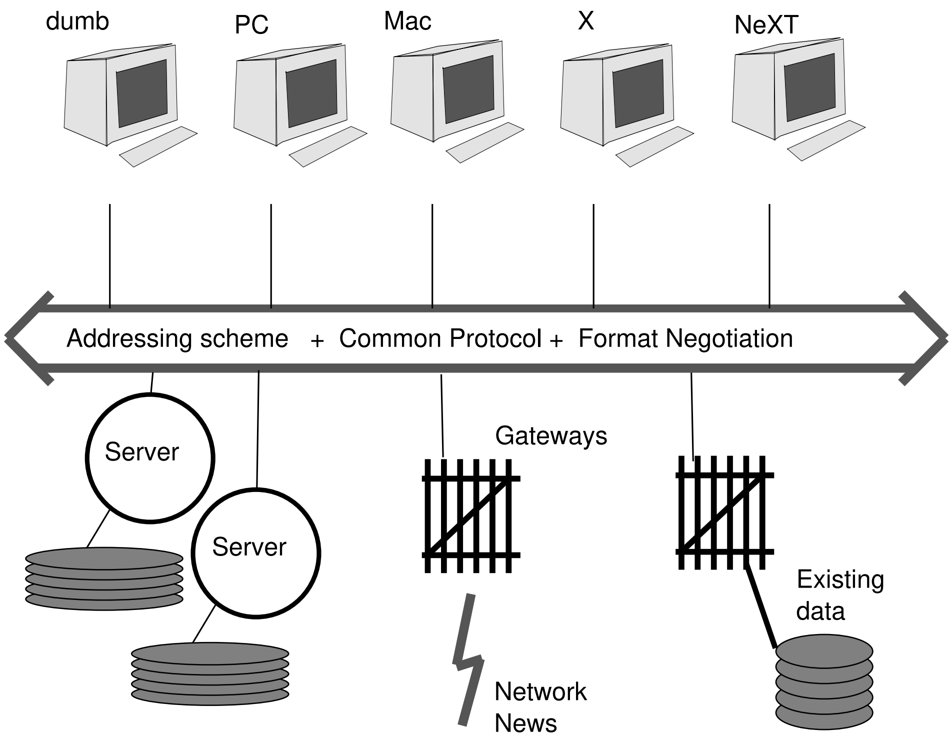

Figure 2.1: The W^3 architecture in outline

2.4 World Wide Web

The World-Wide Web (W^3) initiative aims to construct a global information universe using available technology. [WWW] The project defines a scheme to integrate information from varying data sources. (see Figure 2.1) Similar to the Organization Engine, the model relies on the client-server model for information translation and transfer. Unlike the Organization Engine method, it emphasizes the use of a common addressing scheme and a common information retrieval protocol. Therefore, the integrated data servers must be modified to comply with the established interface.

The W^3 model lacks the flexibility inherent in the Organization Engine approach in which the client application is responsible for the translation task. The approach employed in this thesis facilitates experimentation, and readily adapts to innovative server applications through the use of specific translation drivers. While the W^3 project primarily targets text based information, the Organization Engine may be easily integrated into wider range of information servers, including more structured types of data. For example, in the current prototype development, translation drivers exist for a CODASYL-compliant hospital information system, a MUMPS-based radiology information system, and an SQL database. [OEVDI92].

2.4.1 Hypertext Links and Search Capabilities

The World-Wide Web project incorporates hypertext links and text search capabilities into its wide area networking capabilities. The user can store links to the result of performing a search. These searches are evaluated the next time the user follows the link. In addition, one can store permanent pointers to remote documents using the hypertext-like links. Thus, the user can assemble a personalized network of information by installing links from the local client to remote servers.

The W^3 metaphor of personalized data organization differs from the structured, attribute-value based framework of the Organization Engine. The merits of an organization-based data management scheme are elaborated in Section 3.1. These fielded, attribute based organization capabilities are natural for the incorporation of data from disparate sources and for the personalization of the data to fit the users' needs and preferences. For instance, the hypertext scheme lacks the structure and name-value association inherent in field based data objects.

Chapter 3

Background: Overview Of The Organization Engine And Search Engine

This chapter provides an overall view of the two component systems that are integrated in this thesis. In particular, the architecture of the Organization Engine is described in section 3.1. For the search engine, the client protocol is introduced in section 3.2.

3.1 The Organization Engine

The Organization Engine in its pure form is a software library intended for user interfaces and application programs to manage and manipulate data from disparate sources. This system provides a framework to combine data from different sources through the specification of a common data model. Data from diverse sources are transformed through services into a single, homogeneous representation. The representation chosen to deal with such heterogeneous data is the Self-Describing Objects (SDO) data model.

The goal [of the Organization Engine] is to provide a software base for use in exploring the issues that arise from dealing with vast quantities of data, primarily located in "legacy repositories" – pre-existing systems that merge raw data with structuring information in individual, idiosyncratic ways. [OEVDI92, p. 5]

3.1.1 Self-Describing Objects

The data model chosen for the homogeneous representation is the Self-Describing Objects data model. It is similar to the Object Oriented data model in the sense that the database is composed of data entities, or objects. Thus, in order to access and manipulate the data, an object handle or a reference is needed. "Given this object handle or identifier, it is possible to ask the Organization Engine Tool Kit to provide information about the object's structure. In addition, one can retrieve or modify the component parts of this SDO." [OETR92, p. 4]

Each SDO is composed of an ordered list of fields. Each field has three parts:

Name

A simple string used to distinguish and access the field. Each name must be unique within an object.

Value

This is an arbitrary value associated with a field name. With each value, there is an associated encoding type. Therefore, the value consists of two components, the type identifier and a vector of bits. The set of types is extensible, and depends both on the underlying system and the application implementation.

User Type

This is a simple string that exists solely for the discretionary interpretation of the application programs. In order to provide the application programs with more flexibility over the management of field values, the user specified field type slot is available. The application program may determine the value of the slot and alter it at any point in time. Its purpose is to facilitate the semantic interpretation associated with any field and its value.

The structure of any SDO is independently mutable from the rest of the system. Unlike the object oriented approach where all data entities must be members of a certain class, each SDO is not constrained to any predefined structure. Therefore, the set of fields that denotes an object's structure may be altered at any instance.

3.1.2 Organization of the Data

One of the goals of this software system is to provide personal organization via structuring. The user manipulates the infrastructure of the stored data to suit his or her necessities. This structure is implicit in the relationships between the data entities. A Self-Describing object may hold references to other objects (using a direct pointer or a symbolic link). The method of access to the information is navigation. Thus, one object can function as the root of the object hierarchy. However, there is no stipulation on the number of root objects and whether more than one clique may exist. To retrieve other objects in the repository, the user "navigates" through this framework by following record references. In a sense, objects function as folders, or containers to other objects, in addition to storing more primitive forms of data.

Currently, the user interface of the prototype Organization Engine system is designed around this notion of navigation and browsing rather than search, to locate pertinent information. This concept of organizing data resembles the approach of synthesizing a file system structure. Finding the desired information in this framework is elementary, because the user possesses intimate knowledge of the substructure. This framework is distinct from a file system since it provides a virtual data configuration system on top of the underlying information. Therefore, each user may have a unique, customized view of the data. Yet this approach is not costly in terms of storage, since the user manipulates only the references to the data, rather than the data itself.

However, this data management scheme lacks a more general appeal. It is advantageous to also provide search capabilities over object collections. Thus, an important focus of this thesis, the query links, fuses these complementary search and navigation information retrieval mechanisms.

3.1.3 System Layers

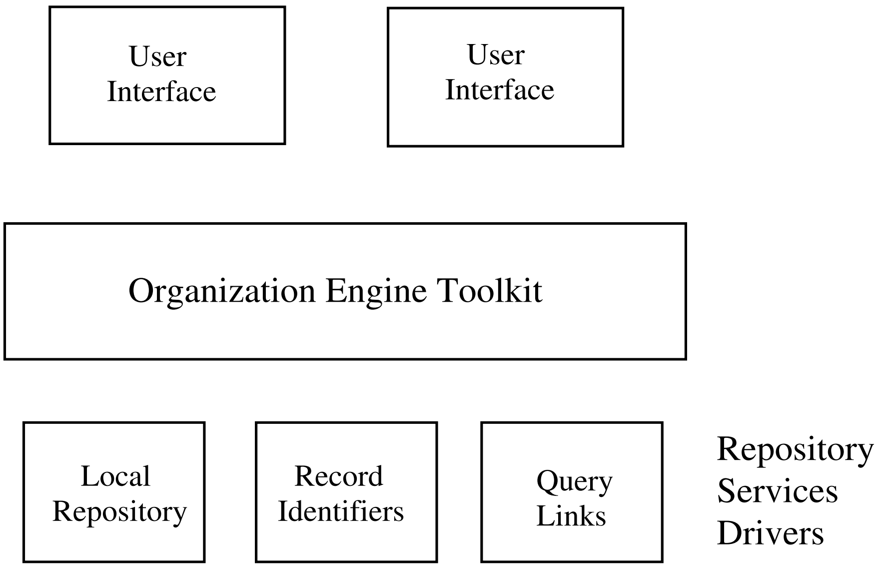

Figure 3.1: Organization Engine System Layers

Figure 3.1 illustrates the three system layers that combine for the complete Organization Engine data management and presentation system. The Organization Engine

Tool Kit is at the heart of the system. It is responsible for the manipulation of the data entities that appear to the User Interface as Self-Describing Objects. The user interface interacts with the Tool Kit through an Application Program Interface (API) The operations include creation and management of data objects using the format of Self-Describing Objects. The Tool Kit communicates with data repositories through the services interface that defines five common procedural entry points for each data source.

User Interface

The Organization Engine Tool Kit is essentially a software library that provides entry points for application programs wishing to manage data. An example of such an application program is a user interface utilized to view and manipulate the underlying data. An important goal in the design of the Tool Kit is to allow for the development of several different user interfaces. This approach insures that the data model function is not restricted to any one method of user interaction. The design allows the set of user interfaces to be easily extended, as research discovers more intuitive approaches.

The User Interface and other application programs communicate with the Tool Kit through an API. As previously implied, this interaction is achieved by using an object reference. The main functions provided by this interface include:

- creation of a new object instance
- retrieval of the structure of an object, i.e. the names of the fields.
- retrieval of any single field in an object using the field name
- manipulation of any single field (changing the value or user type)
- restructuring the object, i.e. replacement of the fields through addition or deletion

Organization Engine Tool Kit

The main component of the system is the Organization Engine Tool Kit. It is in charge of the mediation between the application programs and the repositories, as well as the management of the Self-Describing data entities. As stated above, the User Interface communicates with the Tool Kit through an API using functions concerned with objects' fielded information. The Tool Kit's responsibility is to convert these operations into a "smaller set which communicates back to the actual source of data" [OEVDI92]. The Tool Kit maintains an internal object identifier for each SDO. This identifier is composed of two parts: an identification of its data source , and a unique identifier to be used by the data source. The interaction between the Tool Kit and any data source is achieved by a dispatching to the appropriate data source, and by specifying the identifier for the records.

Data Repository Interface

The purpose of the Organization Engine is to integrate information from disparate data sources into one homogeneous framework. For each information repository there must be a translation driver, or a service. It is essential that the set of services with which the Tool Kit may interface be easily extensible. Hence, there exists a standard interaction protocol between the Tool Kit and any service type. This protocol, which is synchronous in nature, has six entry points from the Tool Kit to the service:

Startup

Notify the data source that the Organization Engine wishes to initiate a session.

Default-Resolve

Given an object identifier, this returns the set of fields in that object. Each field has a name, value and user-type. Since the resolution process may be time consuming, there is an option to change this Default-Resolve procedure. When changing this procedure, the service can store the object information obtained from the data source. This option is particularly useful when the service wishes to cache some of the information. This action is performed to avoid retrieving the data again at a later point during the session.

Modify

Given an object reference and a list of fields, this replaces the contents of the specified object with the new list of fields.

Create

Given a list of fields, this returns the handle for the newly created object or a boolean false if this operation failed.

Modify-Field

Given a object handle, an existing field and a replacement field, this modifies the field.

Shutdown

This entry point notifies the service that the Organization Engine Tool Kit is about to terminate the session.

In this thesis, Modify, Create and Modify-Field were not implemented, because the interface to the Library Search Engine does not currently allow any modification to the underlying data storage from the established connection.

3.2 The Search Engine

"Library 2000" is a project currently undertaken at MIT under the supervision of Professor Saltzer. The goal of the project is to analyze the characteristics and requirements of future electronic libraries. The issues under investigation are diverse and include scalability, presentation client-search engine protocol, value-added indexing, images, reliability, and aspects of a distributed system. This section presents an overview of client interaction with the Library 2000 search engine and data server. The discussion is partially based on the help facilities of the system and partially on the weekly meetings held with Professor Saltzer. [REDRM]

3.2.1 Client Protocol

The client protocol of the search engine is concerned with the formulation of queries over data collections and the retrieval of record contents. The protocol syntax is based on s-expressions. These allow a high degree of flexibility in specifying long and complex queries. There are four distinct functions in the protocol. Three perform searches on data collections. They are FIND, AND, and BUTNOT. The fourth function, GET, accomplishes record content retrieval.

FIND, AND, BUTNOT

The FIND function is used to distinguish a particular set of records that satisfy a certain search criteria. The arguments to the function are regular expressions. The search engine takes each argument. It then calculates the set of records where the argument, a regular expression, appears. Each of these sets is composed of a list of record handles. Finally, it returns the intersection of the sets to the client. Therefore, the FIND function returns a list of unique record identifiers that reference the complete set of records where all the arguments to the FIND function are manifested.

The AND function takes a set of arguments, each conceived as a list of record identifiers. It returns the list of record identifiers that constitute the intersection of these sets. The typical argument to an AND operation is a FIND function specification.

A variation on the AND operation is the BUTNOT. This operation also takes a set of arguments that are lists of record identifiers. It returns the list of record identifiers that appear in the first argument but not in rest of the argument lists. The actual syntax used for this function is:
(AND (FIND "sussman") (NOT (FIND "Mellon")) ...).

GET

The GET operation takes a unique record identifier as an argument. This handle is typically an identifier returned from the issue of a FIND command. The search engine responds with the contents of the identified record. The contents returned currently appear as the full text card catalog record. This string can be parsed into the particular sub-fields that constitute a record. The Organization Engine data translation modules utilize this structured information to construct a Self-Describing object. The set of fields includes a call number, a title, an author, an ISBN number, an LCCN, random information notes, an abstract, key-words and record-keeping information.

3.2.2 Establishing a Connection

The connection utilized in the implementation of the integration is a specially tailored Internet port intended for machine to machine interaction. The port connects an outside application to the search engine. A similar connection, intended for human interaction, can be reached via "telnet reading-room.lcs.mit.edu".

Choosing a Collection

The search based system presently allows query formulation and record retrieval over three distinct record collections. The user may choose any of the three collections through the user interface while selecting the remote database. To the user, the different components appear as distinct databases, although they utilize the same search protocol. The current implementation employs an identical protocol to access the three distinct collections. However, this approach does not appropriately reflect the current methodology of remote database access.

Presently, a user must be familiar with the particular protocol specifications of a remote database if he or she wishes to utilize its services. The syntax and semantics of each collection's search protocol may differ. Therefore, in the current implementation, the specific protocol used for any client-server communication is not transparent to the user of the Organization Engine system. A possible topic for future work is the addition of a translation mechanism from a single, general search language into the specific language used by each data server.

At the server side, the selection of a collection is accomplished on a per-session basis. The client specifies this choice during the communication initialization phase. Once the selection has been declared, the established communications port is used exclusively for query formulation over the particular collection. In order to choose another collection, the client must activate another communications channel. Since the server accepts multiple external connections, the client can simultaneously administer a communications channel for each database selection.

Available Collections

The three collection currently available are:

MIT LCS/AI Reading Room Catalog

This collection includes most proceedings, books and technical reports since 1986. It contains approximately 15,000 items.

MIT BARTON Computer Science Snapshot

This collection contains the MIT BARTON catalog records in the area of computer science as of May 1990. There are currently nearly 16,000 holdings.

MIT LCS & AI Technical Reports

This collection incorporates bibliographic information and abstracts of about 2000 MIT LCS and AI technical reports and memos.

A Stateless Connection

The design of the interaction is based on the notion of a stateless connection. Here, the server does not have to retain any information about its clients or any completed transactions. This form of interaction is preferred for a multitude of reasons, and has been used in various data server applications such as Sun NFS. The rationale for this approach found in the Sun NFS documentation especially applies in the information retrieval environment:

> *The major advantage of a stateless server is robustness in the face of a client, server or network failures. Should a client fail, it is not necessary for a server (or human administrator) to take any action to continue normal operation. Should a server of the network fail, it is only necessary that clients continue to attempt to complete NFS operations until the server or network is fixed. This robustness is especially important in a complex network of heterogeneous systems, many of which are not under the control of a disciplined operations staff and may be running untested systems and/or may be rebooted without warning.* [NFS, p. 15]

Finally, the decision to implement a stateless connection eases the implementation task of the server and possibly the client. It also enables the server to administer a greater number of clients within a given memory space.

Chapter 4

Record Identifiers

This chapter discusses the design and implementation of one approach to the integration between the two systems: the storage and management of unique record identifiers in the Organization Engine. In this approach, the Organization Engine contains references to specific records stored in the Search Engine. These locally stored remote record references can be utilized by the Organization Engine to retrieve record contents from the Search Engine. The Tool Kit is then able to present the record to the application program as a Self-Describing object.

In this chapter, we first introduce the rationale for providing access to remotely stored records through unique record identifiers. Included is the observation that search capabilities are not sufficient to provide the desired function. Section 4.2 then presents an overview of the implementation's architecture.

4.1 Why Unique Record Identifiers?

The maintenance of unique references to remote records allows the client application to repetitively retrieve specific records. The following scenario illustrates this need. During a search session on a database, a user discovers a set of pertinent records. Currently, the user desires to obtain handles on these records to guarantee future access to the information they contain. Unique record identifiers provide this function.

In the above scenario, the user is interested in the set of records that satisfy a particular query at the time of the query formulation. In turn, the user wants to maintain pointers to these records. The query utilized may now be discarded because the set of records has already been obtained.

There are two alternatives to guarantee that the user will be able to access the information from the chosen records in the future. One method involves simply copying the complete record contents into the local store. Section 6.1 elaborates on this option. The other option stores only the reference to the record contents in the local repository. The complete contents can then be obtained upon demand through communication with the remote database. This latter option is the focus of this chapter.

The record identifier access method allows certain important capabilities in information management systems. These capabilities include a guaranteed handle for future access, propagation of record references to other users and applications, reduction of data in local storage and provision of up-to-date information. It also improves the efficiency of information retrieval when the same records are retrieved more than once.

4.1.1 Search Capabilities Alone are not Sufficient

The Z39.50 protocol is an example of an information retrieval protocol that does not allow record retrieval using unique record identifiers. As previously discussed in section 2.2, the Z39.50 protocol requires the data server to maintain state for the client. The state information includes a set of record references that are established through a process of query formulation and refinement. At any point in the session, the client may require the server to supply the full contents of any record currently in the active set.

In this protocol, the only method to retrieve a desired record is to select a particular record out of the active set. The selection is made using an element index. Specifying the same index number at different instances during a session is not likely to result in a reference to the same record. This behavior is a result of a modification in the composition of the active set of records. In addition, there is no method of associating unique record identifiers with data entities in the system (at least in the communication protocol with the client). There is no guaranteed method to retrieve the same record during different sessions.

4.1.2 A Guaranteed Handle for Future Reference

The most important reason a client application may wish to store a unique record identifier is to have a guaranteed handle for future reference. Search capabilities alone do not insure that the client will be able to retrieve a specific desired record at some later date. This behavior is a result of two characteristics of the management of data objects in an information storage and retrieval system: object mutability and addition of matching records.

Object Mutability

The same search specification is not guaranteed to result in the same set of records at two different times. The data objects in an information storage and retrieval system may be mutable. Any part of the record can be changed. Therefore, if text matching is utilized to locate the relevant records, then there is no guarantee that a given query will retrieve the desired record. Unique record identifiers are essential when the client application requires a handle on a particular data entity while allowing the actual contents to change.

Addition of Matching Records

Since the result of the search may vary, the same query may yield more than the desired record reference at a later date. For instance, the initial query may result in a set containing one record reference. The client stores the search specification as a handle. Then, the server adds another record to its database that satisfies the search. The next time the same search query is invoked by the client, the result set consists of two, rather than one, elements. This outcome will result in ambiguity on the client side when the client wishes to retrieve the contents of the previously selected record. The client is unable to distinguish which of the two records is desired. Again, the only method to solve this problem involves maintaining unique identifiers, in addition to the search capabilities.

4.1.3 Storage Requirement

An important aspect of each software system is its storage requirements. To avoid data duplication and reduce local storage, the client application may wish to store only a reference to the record contents. Storing the handle rather than the entire copy of the record greatly reduces the storage requirement. It is unreasonable to expect client applications to store information that can be easily retrieved from the server. In addition, there may not be adequate storage space in the local repository. Finally, the record content duplication results in wasteful data storage.

4.1.4 Propagation of Record Handles

Record identifiers also enable users and application programs to provide others with guaranteed references to remotely located records. With the existence of record identifiers that are understood by others, the flow of information is significantly facilitated.

For example, a document may contain references to other documents or library records. Upon demand, the application program can easily retrieve the full contents of any item referenced, whether it is stored locally or found in a remote database.

The ability to guarantee a reference to a record coupled with repository identification information is essential in a distributed and heterogeneous information retrieval environment. There are currently several efforts, such as the one by WAIS and the World Wide Web to establish a global and unique identifier name space to most of the accessible information on the Internet.

4.1.5 Retrieval using Record IDs may be more efficient than Invoking Search

A more subtle point relates to the efficiency of record retrieval using direct pointers to objects, rather than issuing an elaborate search command. In most implementations of search engines, invoking a search is costly in terms of space and time in comparison to a retrieval of one record using a unique record identifier. When repetitive retrieval of records occurs frequently, this effect may actually have a measurable impact on system performance.

4.1.6 Semantic Meaning of Handle on a Record

A handle on a remotely stored data object contains pertinent information regarding the retrieval process. The information may represent the "road map" used in the execution of query formulation and refinement. In this case, the relevant information describes the process by which a single data object was distinguished from the others. The identity of the specific record found in the process is less important. Here, the client is better served by storing the elaborate procedure employed. Such an approach, which is described later, is implemented in this thesis using the notion of query links (chapter 5).

When the client application is concerned with the actual record found rather than the process used to distinguish it, unique record identifiers are suitable. The client may want to reference and/or retrieve this particular record in the future. The meaning of the handle established between the client and the server is a distinguishing marker, a permanent pointer to some particular data object, i.e. a unique record identifier.

4.1.7 Up-to-date Information

One of the goals of providing retrieval access through record identifiers is to enable the client to easily access the current contents of specific records. This ability is especially significant in domains where the nature of the data is dynamic. In this case, the client maintains a pointer to a specific data entity. The contents of this entity may be continuously changing (such as the current value of a stock). Access through record identifiers is therefore a viable solution to this problem.

A distinction is established here between the time of search and the moment of data retrieval. In the process of finding the pertinent records, the user formulates a query. This action results in a set of references to records that satisfy this search criterion. As previously discussed, at this point, there are two alternatives to guarantee farther access to the information. The user may simply copy the data over to the local repository. In this case, the time of search and the moment of the data retrieval are equivalent and permanent.

The other option, which is the focus of this chapter, makes a distinction between the search and the retrieval phases. The time of search is still permanent (i.e. the set of records represent the set that satisfied some query at a particular point in time). However, the data retrieval occurs upon demand and represents the current contents of the specific records. This process allows the local data management system to continuously reflect the up-to-date information at the remote database.

When the nature of the data is more static, such as in current library systems, having the most up-to-date information is not as important. However, in the case where library records are updated, the identifier becomes an invaluable tool. In the libraries of the future, where patrons may be able to add miscellaneous notes to documents, or where the information in general will be more dynamic, this approach will be highly appealing.

4.2 Overview of the Architecture

In order for the Organization Engine data management system to provide access to remote records using the notion of record identifiers, a new service was added to the data repository interface. This section describes the design and implementation of this service, as well as the addition of several necessary user interface components.

4.2.1 Putting Record References into the Organization Engine

An important aspect of the design of the interface implementation is the introduction of record references into the Organization Engine. The system uses an approach based on query formulation. Through the User Interface, the user selects a remote database and specifies a query to perform in that database. In addition, the user selects the storage location of the record references that are returned as the result of the remote query. One possible location is in a folder inside the local repository.

Identifier Insertion Process

Refer to figure 4.1 for the following step by step description of the identifier insertion process:

1. The user specifies the search and the remote database.

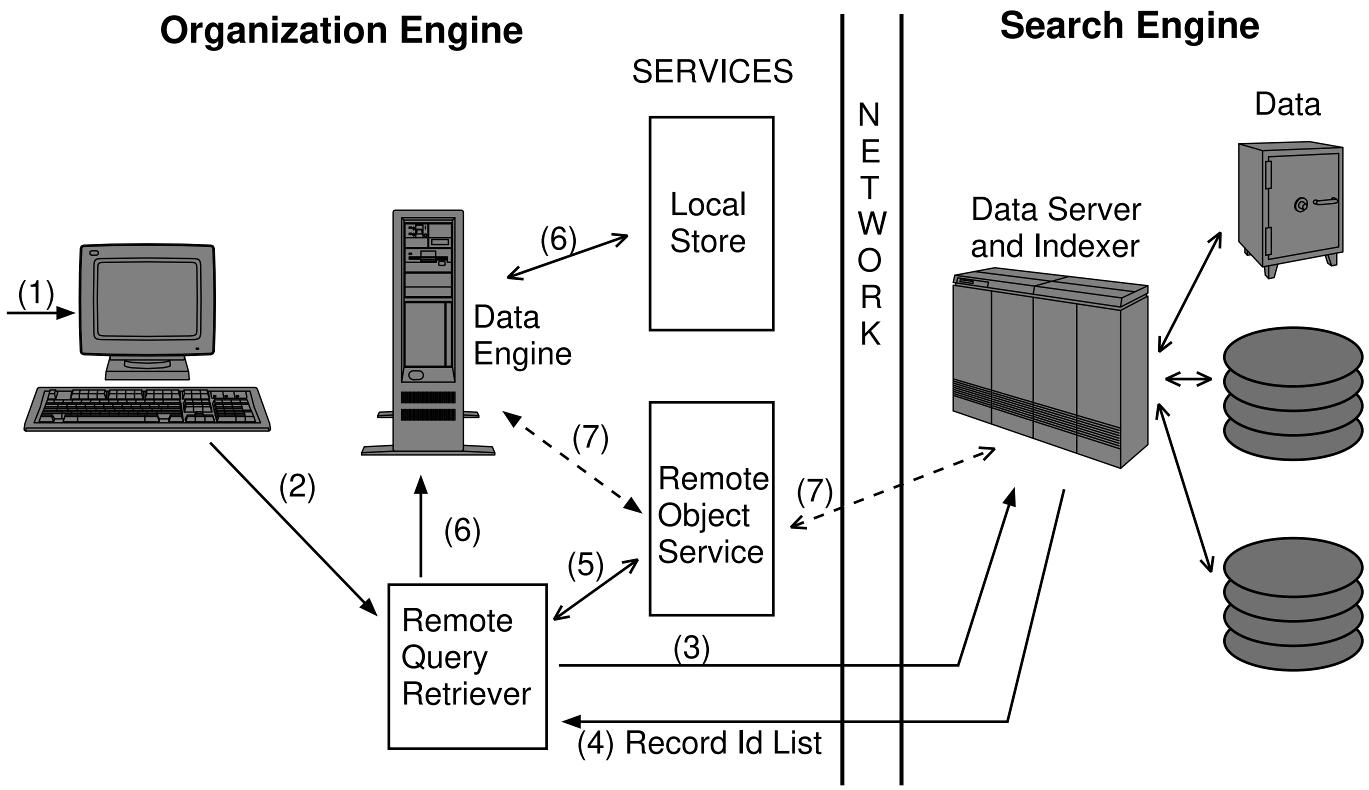

Figure 4.1: Record Identifiers Service Architecture

2. The User Interface passes the query and remote database marker to the Remote Query Retriever (RQR) module, which is essentially a part of the service interface.

3. This module establishes a connection with the specific remote database if a communication channel has not already been instituted. Because of the stateless nature of the communication protocol, a broken connection is inconsequential. When required, the channel is simply re-established. The Remote Query Retriever Module sends the user specified query to the appropriate database engine.

4. In response, the data server and indexer (Library 2000) returns a list of unique record identifiers satisfying the query specification. These identifiers have the necessary properties describes earlier in the chapter.

5. The RQR module parses this list of identifiers and constructs a new record reference for each element of the result set. Using the Organization Engine Tool

Kit, the RQR module then places the new record references in the user specified location.

6. The Organization Engine Tool Kit invokes the Remote Object Service in all subsequent requests for the record information.

4.2.2 The Remote Object Service

The Remote Object Service is a component of the Data Repository Interface layer. It couples an opaque object identifier, supplied by the Library search engine, with the database service marker to create the object handle. To the Organization Engine Tool Kit layer, this object appears equivalent to all other Self-Describing objects.

The most important task of the Remote Object Service is to convert a reference to a remotely located record into the format of a Self-Describing object understood by the Tool Kit. As discussed in section 3.1.3, the interface between the Tool Kit and the service contains six entry points. The resolution entry point accomplishes the translation task from an object identifier to a Self-Describing object.

The arguments passed into the resolution procedure are the remote object identifier, a possibly *null* change procedure, and a possibly *null* new object contents. The object identifier is a unique record identifier (in this case, a record number). This identifier is the argument to a GET record request to the search engine (for GET syntax see Section 3.2.1). The search engine returns a character string to represent the record contents. The Tool Kit parses this result string into a list of fields. It then utilizes the list of fields to construct a Self-Describing object. The second argument to the resolution procedure implements the caching scheme, a topic discussed in the next section. The last argument is not used in this implementation. It allows the Tool Kit to modify the contents of a record to a new list of fields. Currently, however, client applications to the Library engine are not allowed to manipulate the stored data.

4.2.3 Storing Record Information

An advantage of using references to records, rather than copying and storing the data, is the drastic reduction in local storage requirement . However, when the information is initially requested by the application or the user, the Remote Object Service must fetch the data. It will then temporarily store the data. Significantly, the user may request access to this particular record more than once during a session.

Repetitive retrieval from a remote data source may be costly in terms of response time, total network usage, and possibly money (in the case where the data source charges the client for each retrieval request). Therefore, a (possibly incoherent) cache is used to store the information. The current implementation of the cache is transparent to the Tool Kit and the application program. It conserves the information on a per-session basis. Thus, the first time the user accesses a record in a session, the cache is updated with the current data from the remote repository. The cache is consulted for the record information on subsequent record inquiries. The implementation utilizes a form of *lazy evaluation*, where the information is not acquired until a need for it is established.

An implementation of a production level information retrieval system must consider the issues of copyrighting. The current implementation ignores the surrounding issues by assuming that once a client obtains a copy of a record, the client application may utilize it in any fashion it desires. However, a different contract between the server and the client may introduce copyrights and specify a cost for each use of the document. In this scenario, the above mentioned caching scheme could not be utilized without some modifications that incorporate cost accounting.

Another consideration is that the cache may possibly represent out-of-date information to the Tool Kit. Since the Tool Kit stores the record contents for the duration of the session, the cache is not updated to reflect the contents of the remote record. Therefore, the implementation does not currently guarantee that the information stored locally will precisely represent the information at the remote data source. This caching scheme variance may therefore be termed *per-session caching*.

Efficiency and Accuracy of the Per-Session Caching Scheme

The caching scheme implemented has proved to be efficient in the development environment because of the following presumed characteristics of system usage:

- With respect to storage requirements, during any one particular session the user accesses only a small fraction of the records. Therefore, locally maintaining a copy of the data does not overburden the local store.

- With respect to system performance, it is likely that a record accessed once during a session will be subsequently accessed during that session. With caching present, successive record accesses are significantly faster and reduce overall network usage.

- With respect to the currency of the information, a session is assumed to persist no longer than several hours. The user may require the information in the client to accurately represent the up-to-date information in the remote database. The per-session caching approach does not unequivocally guarantee that the information in the local system precisely reflects the data in the server. However, because of the static nature of the data in library information retrieval systems, the performance is satisfactory. The cumulative detrimental effect of inaccuracies in this system is predictably negligible when compared to the benefits achieved with per-session caching. This small detrimental effect is a result of the minuscule probability of an error. Incoherence in the cache may exist only in the situation where the search engine mutates the record contents while a copy of the record is stored in the cache. Since the Tool Kit presumably maintains the cache for a period of no longer than several hours, and record contents updates are infrequent in the search engine, incoherence will occur only sporadically. Therefore, the per-session caching scheme will rarely falter. Because of the efficiency gains in access time to record contents, such errors can presumably be tolerated in the current user access metaphor. For connections with other types of systems, this caching scheme may not be appropriate since the data may be more dynamic than information stored in current library systems.

Extending the Notion of the Cache

Idealistically, the user should be able to control some of the cache's characteristics. The desired form of the cache in this system is therefore not genuinely equivalent to the prevalent notion of caching. The caching process should not be completely transparent to the user or the application program. Two attributes that the user may desire to control are how often and under what conditions the cache is updated. For instance, the user may require the information present in the system to be considerably current. Therefore, on every access to any remote record, the Remote Object Service will have to fetch the information from the remote database. Other options include cache information update after a pre-specified time interval, or an update that is dependent upon the length of time the record has been in the cache. This thesis does not investigate any of the above extensions, although further research may yield beneficial results.

Chapter 5

Query Links

The integration between the Organization Engine and "Library 2000" can be implemented by the use of query links. This chapter discusses the storage and management of query links in the Organization Engine. In this approach, the Organization Engine manipulates data objects that store search specifications. The query link service utilizes the search specification embodied in the data object to construct references to remote records. Thus, a query link object contains references to remote records handled by the unique record identifier service as discussed in chapter 4.

In this chapter, we first clarify the definition of a query link. Section 5.2 reveals the significance of maintaining query links. We continue by discussing the semantics of the object identifier and conclude by providing the overview of the system architecture.

5.1 What is a Query Link?

The query link handle contains a search specification. The query link service utilizes this specification to retrieve the set of record references that match the search. Fundamentally, the query link is a folder that contains these records. Each of the pinpointed records is governed by the unique record identifier service.

Significantly, objects contained in a query link are ephemeral. Because the record references may unexpectedly disappear, a user may request a guarantee for future reference to a pertinent record. Therefore, record references in a query link can be copied or moved to a different location in the local store. This process insures that the record references become a permanent constituent of the user's repository.

5.2 Why Query Links?

The maintenance of query links enables the client to perform repetitive searches. This approach allows the user to preserve and repetitively employ a particular search. The search specification provides a "road map", or a procedural description, of the method to locate references to pertinent records. In this technique, the process of filtering information is of importance. To preserve the filtering process, query links can be permanently stored in the local repository.

5.2.1 Concurrent Query Processes

The storage of query links in an organization-based repository presents an extension to the system. In addition, it alleviates problems that exist in systems based solely on search capabilities. For instance, a user of a search engine interface will most likely require that the application will preserve the user's previous transactions. The satisfaction of this requirement may involve the maintenance of an active set of record references, or the preservation of a search request that the user continuously modifies and refines. Query links, as well as most implementations of search engine clients, provide these functions.

However, user interaction with a search engine should not be limited to the above mentioned constrained function. A researcher currently locating background literature may desire to investigate several issues in parallel. Therefore, the client application should support several simultaneous query endeavors. In addition, it would be advantageous if the queries were stored in between sessions, to allow an effortless continuation of the research process. The inclusion of query links in the Organization Engine provides exactly these functions.

5.2.2 Up-to-date Search Result

Query links provide a different alternative to "the time of search and moment of retrieval" quandary than unique record identifiers or immediate duplication implementations. For query links, the time of search is the present. Hence, the set of record references contained in the query link reflects the up-to-date search result embodied in the link. Once the set of references has been established, the user may retrieve any element in the set. Although the search precedes the data retrieval phase, the time of search and the moment of retrieval are essentially equivalent.

In an environment where the data is dynamic and search results change frequently, query links are undoubtedly useful. However, even in an environment where the data is fairly static, such as in a library application, query links are beneficial. The storage of query links in an organization-based environment allows the researcher to perform the same search in the present and the future without the re-specification of the query. The organized repository allows the researcher to conveniently store and subsequently locate and retrieve the pertinent queries. This process is accomplished effortlessly using the query link service.

5.3 Caching Revisited

The caching scheme utilized for the implementation of query links is similar to the method used for record identifiers discussed in section 4.2.3. The search result, a set of record references, is stored in the query link service on a per-session basis. Again, lazy evaluation is used. The service communicates with the remote database only when the application program requests the contents of the query link. The result is then cached in the local store for the duration of the session.

5.4 Semantics of the Object Identifier

The object identifier used for administering the query links is composed of two elements. The first component distinguishes the remote database. The second element is the query specification. In the interaction with Library 2000, a string in the format of a FIND request is incorporated (see Section 3.2.1). Upon demand from the Tool Kit, the query link service communicates this string to the search engine interface. The service consults the response, which is a set of record identifiers, to construct the record references within the query link.

5.4.1 Creating a Query Link

Communication with the data server is not necessary in order to construct a new query link object. The Tool Kit simply constructs an object identifier that specifies the query and identifies the remote database. The Organization Engine retrieves the list of record identifiers that match the search only upon user demand. This process contrasts with the record identifier method. There, the Tool Kit invokes a remote search to obtain the list of record identifiers immediately after the user specifies the complete query.

The origin of the object identifier dramatically differs from the origin of all other identifiers in the Organization Engine system. For other objects, the lower layer is always responsible for assigning the unique record identifier. [OETR92] In the query method, the user who creates the query link assigns the object identity through the selection of the remote database and the search string.

5.4.2 Editing Query Links Identifiers

The process of query modification and refinement is facilitated by the query link identifier editing capability. Once a query link has been created, the user may alter its search specification or the remote database selection. The content of the query link are then adjusted to reflect the new result of the search specification. This interaction can be used analogously to the search refinement process in other information retrieval client applications. Again, the distinction is that the query links are permanently stored in the local repository.

5.5 Architecture Overview

A new service is incorporated to the Organization Engine data repository interface in order to provide query links. The service relies on the Remote Object Service discussed in section 4.2.2 for the management of the unique record identifier references.

5.5.1 Query Link Creation and Management

Figure 5.1 provides an overview of the system architecture and module interaction:

1. The user specifies the search and the remote database. The user interface transmits the information to the Organization Engine Tool Kit.

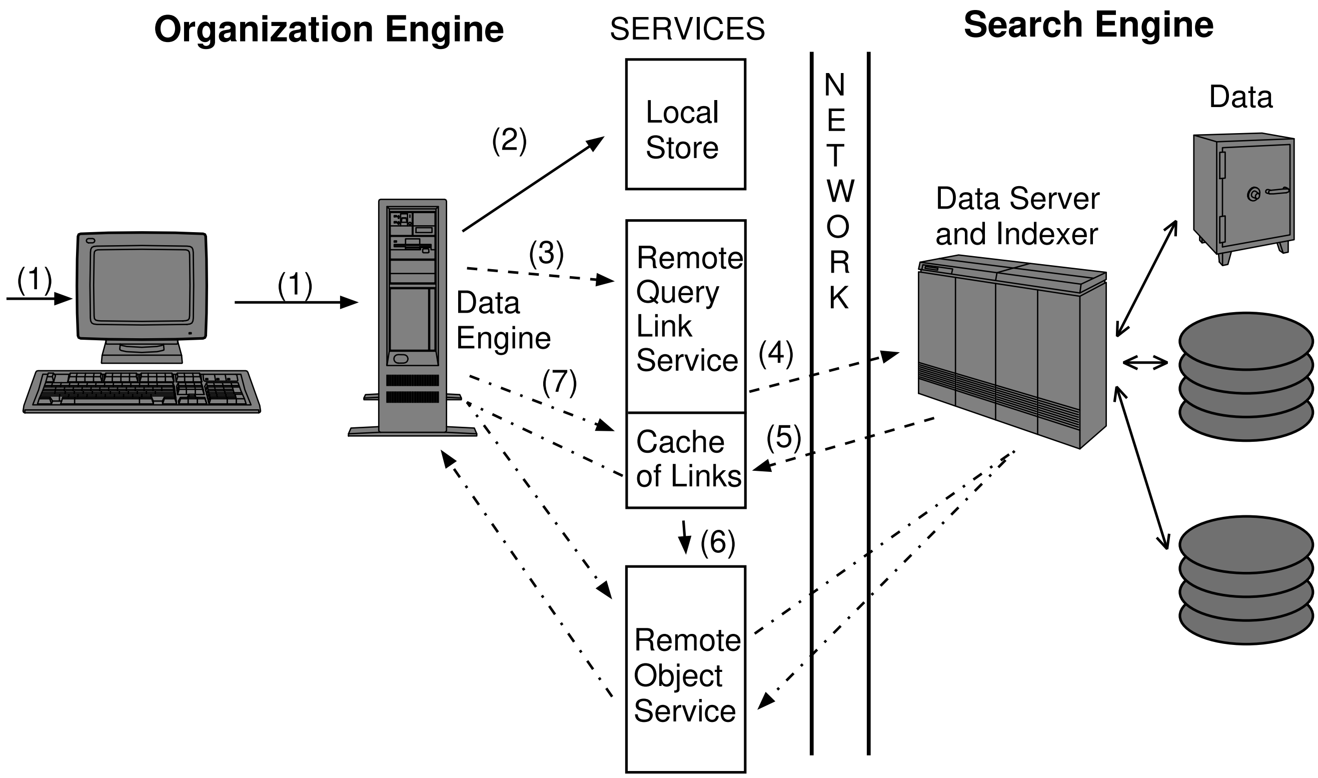

Figure 5.1: Query Link System Architecture

2. The Tool Kit creates an instance of a query link object that embodies the user's search specification. A reference to this object can be placed in the local repository to enable future access to the query link. It is significant to note that the creation of a query link object does not necessitate interaction with the remote search engine.

3. When the contents of the query link is requested, the Tool Kit consults the Remote Query Link Service (RQLS) module.

4. The RQLS module maintains a local cache of query results. If the content of the object is not in the cache, the RQLS module invokes the remote query on the specified database. The query result must be a set of record references. In the case of Library 2000, the query should have the format "(FIND ...)", or a variant discussed in section 3.2.

5. Library 2000 returns the set of unique record identifiers that reference the records satisfying the search requirement. The result is identical to the one received by the Remote Query Retriever Module (the module used for the record identifier service).

6. The query link cache is updated with the current list of record references. The RQLS module communicates the required information to the Remote Object Service in order to construct new remote record references that can be manipulated by the Tool Kit, .

7. When the Tool kit requires the content of a remote record reference found in a query link object, it consults both the RQLS module and the Remote Object Service module. The Remote Object Service module retrieves the up-to-date record information in a method similar to the one used to retrieve other remote record contents.

5.5.2 Copying Record from Query Links

One of the goals of query links is to locate references to pertinent records. After locating such records, the user may desire to maintain permanent handles on these pertinent records. Since the record references in the query links are ephemeral, the references in the query links may disappear in the future. This phenomenon can result from a modification of the query embodied in the link, or because of a mutation in the remote object. Therefore, to ensure permanence, the query links service allows the user to copy or move record references from the query link object to other locations in the local repository.

Chapter 6

Other Options

There exists other alternatives for the integration of the two data management systems. These options include immediate duplication, partial storage and browsing of the search indices. This chapter outlines the fundamental issues of each of these possibilities that have not yet been implemented.

6.1 Immediate Duplication

This section describes a relatively elementary approach to the integration of the two systems: immediate duplication. In this method, the user specifies a query that is then communicated to the search engine. The contents of all records that satisfy the search criterion are subsequently copied to the local repository. A new Self-Describing object is then constructed to correspond with each element in the result set. Thus, the information replicated from the remote database becomes an integral part of the local repository.

6.1.1 Why Immediate Duplication?

In addition to the relative simplicity of immediate duplication, other rationale exists for implementing this approach. For example, the user may desire to take a snapshot of the information at the remote database. Here, the user requires a copy of the information as was discovered in the search engine during the initial search and retrieval phase. There is no requirement that the data precisely reflect the current information available at the data server. The information, as found at the moment of search, is adequate. Furthermore, the replication of data to the local repository guarantees that the user will possess an access path to the pertinent information in the future.

Guaranteed Future Access

Maintaining a handle on a remote record does not guarantee future access to the pertinent information. A user may be particularly interested in the information stored in some remote record. Under three circumstances, a unique handle will not certify subsequent access to the pertinent information contained in the record. First, the record may be omitted from the remote data server by the curator of the database for various, and possibly unknown reasons. This dilemma does not constitute a difficulty if the remote source never deletes records from its repository, or always assures future access to the information in some other fashion. For example, if the record is deleted, the data server may provide directions for the retrieval of the record from another location.

Second, even in an environment where access to the records is certified with the use of a permanent unique identifier, the pertinent information may be lost. If a record is modified, information pertinent to the user may be deleted or profoundly altered. Therefore, in an environment where remote records are mutable, the exclusive method to guarantee future access to the pertinent information is the replication of data in the local store.

Finally, the location of a remote record depreciates the reliably of information retrieval as compared to local storage of the data. Distributed systems are clearly prone to faults, because of network or remote data source malfunction. The storage of information locally provides an alternative attempt to increase the reliability of information access.

6.1.2 Modification of Records

In contrast to the above scenario, the user may desire to alter the information in a record. It is therefore advantageous to store a copy of the record in the local repository. The user is now able to modify the information in the record. For instance, the user may add miscellaneous notes or delete fields that are malapropos. However, the general copy available to other clients is unaffected. Therefore, the local copy of the record is transformed to suit the user's provisions, while other clients of the search engine are not afflicted.

In addition, the current implementation of the search engine does not allow modification to the database from the client interface. It is clear that only privileged clients should be awarded this capability. Therefore, for clients without modification authority, the immediate duplication method compensates by enabling the client to refashion their own local replica rather than the global representation.

6.1.3 Storage Considerations

A prominent disadvantage of the current approach is data duplication. Storage requirements are increased because of the acquisition of complete records in the local repository. The number of records stored locally may be manageable, because the user stores only records that are pertinent. It is plausible that the fraction of records in the data server pertinent to any particular user is rather insignificant. However, the precise

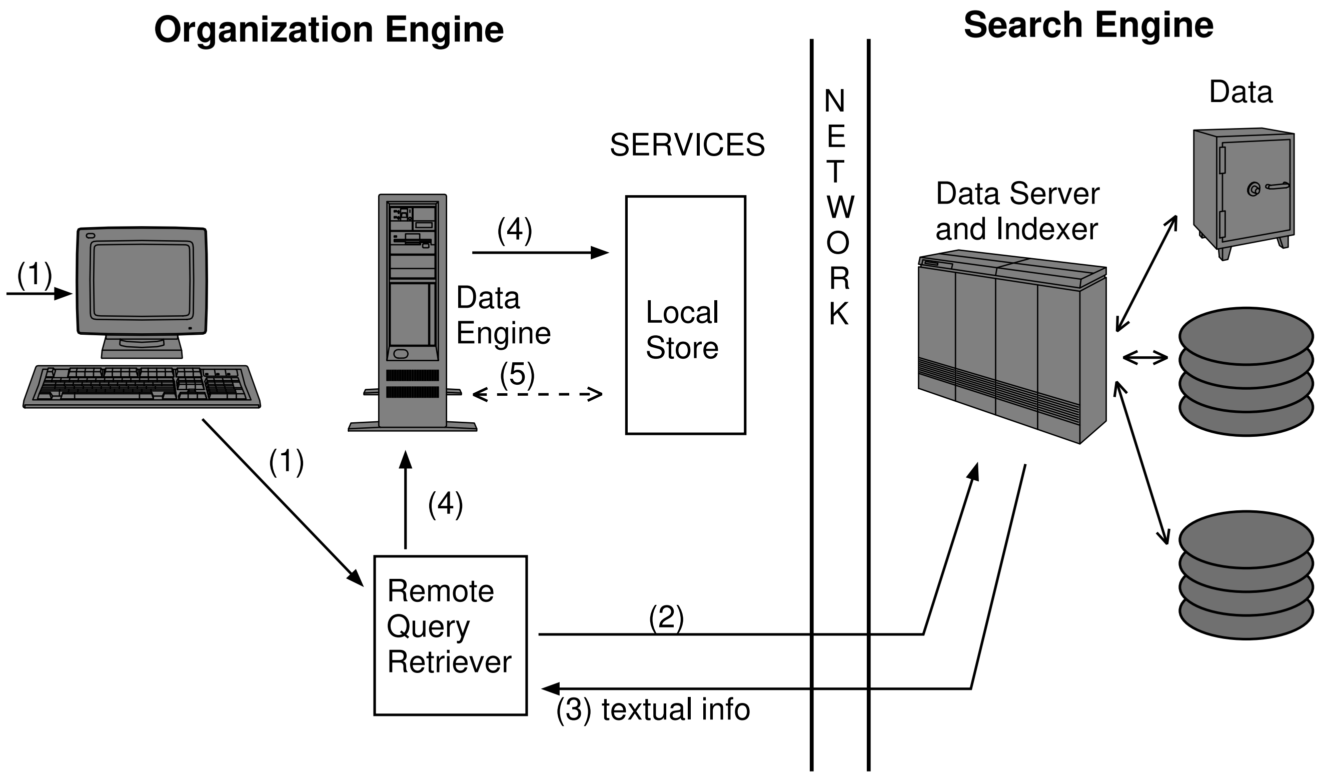

Figure 6.1: Immediate Duplication System Architecture

burden depends on the characteristics of the information stored locally. For example, the accumulation of even a small number of full manuscript PostScript images can prove overwhelming to the local store.

6.1.4 Architecture Overview

In order to provide immediate duplication capabilities in the Organization Engine, it is not necessary to incorporate another service in the set of data repository drivers. The only additional module required is an enhanced version of the Remote Query Retriever (RQR) module utilized in implementing the unique record identifier service (see section 4.2). The RQR module simply parses the record contents transmitted from the search engine. It constructs new records that embody the information in the remote record, and places them in the local repository.

Refer to figure 6.1 for the following step by step description of a possible implementation architecture for the search and duplication process:

1. User specifies a search and a remote database. This information is communicated to the Remote Query Retriever (RQR) module.

2. The RQR module dispatches a query to the remote database. This query is of the form "(GET ...)", where the argument can be either a search specification or an actual record identifier.

3. The search engine responds with the record contents of all records satisfying the search criterion.

4. The RQR module parses each record contents (returned by the search engine) into a list of fields. It then constructs new Self-Describing objects in the local store using these lists of fields.

5. This new information becomes an integral part of the local store. There is no more communication with the remote database concerning the records presently in the local store.

6.2 Partial storage

It may be advantageous to implement a system where only some components of the remote records are stored locally. Here, the user specifies which parts of the record should be updated from the remote database, and which parts remain under the management of the local repository. This preference constitutes the primary rationale for an integration based on the partial storage method. Contrary to immediate duplication, changes to certain components of remote records are propagated to the local store.

6.2.1 Why Partial Storage?

Partial storage of record contents has several practical applications. First, by locally storing a subset of the information under client supervision, the user is able to modify and manipulate certain components of the record contents. In addition, the user can also attach miscellaneous fields to the record. However, the other components of the record reflect the current information in the remote database.

An implementation of partial storage may lead to the following interesting scenario. Suppose several clients cooperate in the management of a particular record. Each client governs a specific component of the record. Therefore, the copy in the data server is, in effect, a conglomerate of all the local copies. Issues such as access control to the records, retrieval and update conflict resolution, and record contents coherency can be the topic of further research in this area.

Storage Considerations

A straightforward practical application of the partial storage approach is an attempt to reduce local storage and data duplication. The client may elect to locally store only those components that are not sizable. Thus, while the name of an image and other relevant information is stored locally, the actual image may be stored in the remote database. The tradeoff then becomes one of access time versus space requirements. Local storage furnishes short access time, but is expensive in terms of client space requirements. The characteristics are reversed for the maintenance of links to record component.

An important design issue is user control versus the application authority regarding which constituents of the record should be managed locally. Is it more advantageous to allow the user complete jurisdiction, or is it more appropriate to authorize the application to determine this behavior? For instance, the client application may decide to store information locally, based solely on the size of the component, while the user would be more concerned with access time.

6.2.2 Implementation Ideas

In the implementation of partial storage, the client must be able to sift through the record contents retrieved from the remote database, and update the local replica only in specific fields. An extension to the search engine protocol could aid in this context. The enhancement will allow the client application to specify a set of fields within a record that it wants to retrieve. This enhancement will both aid the client (or a human user of the search engine) to filter the relevant record contents, and also reduce network traffic.

6.3 Browsing / Traversal through the Search Indices and Data Categories

An important element of information retrieval in an organization-based repository is search by association or other groupings. These groupings cannot necessarily be clearly defined by the user. An example of the search activity is browsing through the search indices or other data categorizations. This activity cannot be easily accomplished using search specifications.

The capability to traverse the access indices of a search engine utilizing the Organization Engine interface is advantageous. In addition, the browsing through value added data categorization is beneficial. The following subsections discuss why these methods facilitate information discovery.

6.3.1 Why Traversal of the Search Indices?

The search indices of the data server hold valuable information beyond their direct utilization by the search engine. In actuality, the search engine merely functions as an intermediary between the user and the search indices. Therefore, allowing the user to browse through the indices while also performing set operations such as AND, OR, and BUTNOT may improve the utility of a search engine. For instance, if the user is not certain of the spelling of an author's name, he or she may benefit from this capacity. Here, traversing through an alphabetical author name list may be preferable to the derivation an appropriate query.

In this framework, the user traverses the search indices and collects the pertinent record references. Simultaneously, the user can perform set operations to construct a working group of the desired record references. When needed, the user can also incorporate search results that cannot be easily achieved with traversal of the indices (such as certain wild-card queries). In the context of a search engine where the full text of the records is indexed, and no structural information exist, such a feature could be a very useful.

6.3.2 Why Value Added Data Categorizations?

The categorization of data contains substantial value within itself. A service that provides access to data groupings can greatly facilitate the search phase of information retrieval. An example of such a facility is the Library of Congress publications categorizations. The typical library system user frequently utilizes these categorizations to find references to pertinent records. The record location method illustrates the benefits provided by a value-added indexing service. The navigation based user interface of the Organization Engine is an obvious candidate to function as the interface between this service and the local system.

6.3.3 Implementation Considerations

The user interface of the Organization Engine and the above approaches are both based on navigation to locate information. Therefore, the implementation of these approaches as additional data repository services is relatively straightforward because of the inherent structure in the indices.

However, there are several important considerations in the design of a search engine system that provides these capabilities. First, the search engine must supply indexed information based on a fielded structure of the records. However, whether to supply this function on all the fields or only on a selected group is an application specific consideration. In addition, in the full text implementation of such a service, should an index exist for each field, or should there be additional fielded information in the current indexing scheme? Moreover, in the interaction with the client, the search engine can reveal the search indices through a special service. Alternatively, one of the records can function as an index. An index record can be administered and updated regularly by the search engine to reflect the current repository status. The Organization Engine may utilize the unique record identifier service to maintain a handle on this special data entity.

Chapter 7

Conclusion

This thesis presented a prototype integration of the Organization Engine and "Library 2000". The implementation successfully incorporates search capabilities into an organization-based framework. Specifically, two access facilities utilizing the notions of query links and unique record identifiers were added to the set of translation drivers for the Organization Engine Tool Kit. In addition, the user interface of the Organization Engine system was enhanced to accommodate these added functions. First, this chapter examines the merits of the two access facilities implemented for this thesis. Then, the discussion suggests some relevant future work.

7.1 Insights

Unlike other information retrieval methods, the unique record identifier method successfully satisfies significant requirements of information retrieval systems. In order for the Organization Engine to incorporate data from disparate sources, it must be able to access search based data servers in a coherent fashion. The unique record identifier method allows the user to store permanent references to pertinent records. Thus, the user is able to repetitively retrieve the possibly changing contents of specific records.

As a result, record handles may be propagated to other users of similar information systems. In addition, the record identifier method presents up-to-date record contents to the user. An important issue here, which merits future investigation, is the design of an appropriate information caching scheme to fit the implementation's environment. It is observed that the per-session caching scheme currently employed satisfactorily addresses the issues of storage requirement, access time requirement, and currency requirement. However, it does so only within the context of servers that manage static data.

This thesis discussed the importance of maintaining unique record identifiers in the hope that all future implementations of information retrieval systems provide this data access ability. The record identifiers' effect is significant in any situation where the user accesses the same database more than once with regards to the same topic.

The storage of record references within an organization-based repository appears natural and advantageous. The organized store allows the user to freely and efficiently construct groupings of pertinent records. These groupings facilitate any future information location and retrieval. The scenario works in the same manner that storing files in a tree structure organizes and formats a user's information categorization. Locating information is trivial when the user has the personalized, intimate knowledge of the substructure. Therefore, the storage of remote record references within the Organization Engine is essential in contemporary global information environment.

The implementation of query links was largely facilitated by the existence of the code for handling record identifiers. It is observed that query links are beneficial in the information filtering process. The query link concept enables the users to engage in concurrent and persistent query processes, as well as to maintain up-to-date search results for queries utilized more than once. The user interface capabilities to create and edit query link identifiers permit the above functions.

Other systems, such as the Semantic File System, Wide Area Information Servers, and World Wide Web, have used a similar concept of storing a search specification for repetitive use. However, it is only the implementation of such a concept in an organization-based environment that fully exploits this capability. Because the user may copy record references stored in query links, he or she may guarantee future access to records that are pertinent.

The other alternatives discussed in chapter 6 for the integration between the two systems may also be beneficial for future information retrieval implementations. The immediate duplication method guarantees quick access to information, at the cost of greater local storage, duplication of data, and information becoming out-of-date. The partial storage concept solves these problems by storing selected components of the information locally, while maintaining pointers to the other constituents. Finally, the browsing method allows another useful form of associative access to data. It is apparent that these wide-area network capabilities are essential in the contemporary global information environment.

7.2 Future Work

This thesis reveals a path for the exploration of modern information management and retrieval systems. The research path focuses on the incorporation of data from disparate sources, and the ability to access distributed and remotely available information systems. The insights gained from this thesis point to some possible future work in the area:

- Users of the Organization Engine will benefit from the development of more translation drivers. Services to incorporate WAIS and W^3 will further exploit the potential to combine legacy data, an attribute inherent in organization-based systems. The implementation of these drivers requires a relatively small amount of work while adding access to a large number of data sources.

- The other concepts to incorporate search based and organization-based systems discussed in chapter 6 can be implemented.

- In order to incorporate other types of data sources, a more thorough analysis of the caching requirements and strategies for query link and record identifier is required.

- To simplify the user task of accessing diverse data repositories, a translation mechanism should be added from one single search syntax to each particular server protocol. If the remote data sources have the same search capabilities, but a different syntax, user transparency to the syntax is preferred. However, for fundamentally different data sources, such as an object-oriented data base versus a full-text information retrieval system, user level transparency to the search protocol is not advantageous.

- A more intuitive user interface to the Organization Engine (for instance *Maya 2000 Points Of Light* [JSM92]) should be implemented. This enhancement should include a different metaphor for the incorporation of search abilities. [KSW92]

- In order to exploit the benefits of the client application to the overall system, the implementation needs to provide the ability for data updates initiated by the client. The data manipulation should transpire within the context of the organized repository.

- The client should be able to incorporate results from multiple data sources that include the conciliation of numerous result sets and multiple references to identical records. The same bibliographic record may be stored in different data collections, yet represent the same physical entity. In addition, different bibliographic records may reference the same book, yet in a different language, or from a different publisher (i.e. one in the U.S. and one in England).

- Further investigation of the contract between the two systems would be advantageous. For instance, what are the implications if records are guaranteed only for a prespecified time? Here, the record identifier may need to include expiration information. [JHS92]

- In order to enhance the Organization Engine, personalized views should be made accessible to other users upon demand. This requirement may include the transfer of record identifiers and query links to others. Ultimately, the Organization Engine could function as a value added index server.

This thesis has continued to research the distributed information retrieval and management environment. While the proposed access facilities provide useful capabilities within this framework, they merely introduce the relevant issues. However, I believe that this thesis can serve as another stepping stone of research in making global information more readily accessible to computer users.

Bibliography

[Z39.50] "American National Standard for Information Retrieval Service Definition and Protocol Specification for Library Applications", ANSI/NISO Z39.50-1988, ISSN:1041-5653, Transaction Publishers, New Brunswick, USA

[WWW] Berners-Lee, T., et al. (1992), "World-Wide Web: The Information Universe", *Electronic Networking: Research, Applications, and Policy*, Vol 1, No 2, Meckler, Westport CT, Spring 1992

[SFS] Gifford, D.K., Jouvelot, P., Sheldon, M.A., O'Toole, J.W. Jr, "Semantic File Systems", ACM 0-89791-447-3-91/0009/0016, 1991

[KAHL89] Kahle, B., "Wide Area Information Server Concepts", Version 4, Draft, Thinking Machines Coprporation, October, 1989,

[KAHL90] Kahle, B., et al "WAIS Interface Prototype Functional Specification", Thinking Machines Corporation, April 1990

[OEVDI92] Miller, J.S., Neidner, C., London, J., "The Organization Engine: Virtual Data Integration", Technical Report CRL 92/3, Digital Equipment Corporation, Cambridge Research Lab, 1992

[OETR92] Miller, J.S., "The Organization Engine", in draft

[JSM92] Miller, J.S., personal communications

[NFS] "Networking on the Sun Workstation", Sun Microsystems, Part #800-1324-03, Revision B of 17, Feb. 1986

[REDRM] Reading Room help facility, "Telnet reading-room.lcs.mit.edu"

[JHS92] Saltzer, J.H., personal communications

[KSW92] Winer, K.S., personal communications