Content Routing: A Scalable Architecture for Network-Based Information Discovery

by

Mark A. Sheldon

B.S., Duke University (1984)
S.M., Massachusetts Institute of Technology (1990)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

December 1995

Signature of Author .....
Department of Electrical Engineering and Computer Science
11 October 1995

Certified by .....
David K. Gifford
Professor of Computer Science
Thesis Supervisor

Accepted by .....
Frederic R. Morgenthaler
Chairman, Departmental Committee on Graduate Students

Content Routing: A Scalable Architecture for Network-Based Information Discovery

by

Mark A. Sheldon

Submitted to the Department of Electrical Engineering and
Computer Science
on 11 October 1995, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

Abstract

This thesis presents a new architecture for information discovery based on a hierarchy of *content routers* that provide both browsing and search services to end users. Content routers catalog information servers, which may in turn be other content routers. The resulting hierarchy of content routers and leaf servers provides a rich set of services to end users for locating information, including *query refinement* and *query routing*. Query refinement helps a user improve a query fragment to describe the user's interests more precisely. Once a query has been refined and describes a manageable result set, query routing automatically forwards the query to relevant servers. These services make use of succinct descriptions of server contents called *content labels*. A unique contribution of this research is the demonstration of a scalable discovery architecture based on a hierarchical approach to routing.

Results from four prototype content routing systems, based on both file system (NFS) and information system (HTTP) protocols, illustrate the practicality and utility of the content routing architecture. The experimental systems built using these prototypes provide access to local databases and file systems as well as over 500 Internet WAIS servers. Experimental work shows that system supported query refinement services provide users with an essential tool to manage the large number of matches that naive queries routinely generate in a large information system. Experience with the prototype systems suggests that the content routing architecture may scale to very large networks.

Thesis Supervisor: David K. Gifford
Title: Professor of Computer Science

Acknowledgments

During my graduate career, I have been blessed with many friends and colleagues who have contributed greatly to my work and well-being.

My advisor, David Gifford, has been very patient and was instrumental in helping me get started on this project. He pushed me and made many suggestions when I thought I was completely stuck. My thesis committee members, Jerome Saltzer and Barbara Liskov, also provided useful feedback.

I am indebted to Programming Systems Research Group members past and present for their assistance and friendship. Franklyn Turbak, Andrzej Duda, Pierre Jouvelot, and Melba Jezierski read thesis drafts and gave valuable comments. Andrzej Duda, Brian Reistad, and Jim O'Toole contributed ideas and code to the project. Pierre Jouvelot contributed various collaborations and many airplane rides. My officemate, Ron "Wind Beneath my Wings" Weiss, has added considerably to this work and to ongoing research. Useful discussions and comments were numerous, but I would like to mention Jonathan Rees, Mark S. Day, Bien Vélez, Chanathip Namprempre, Pete Szilagyi, and Michael Blair. Ken Moody and Jean Bacon of Cambridge University made many useful suggestions, some of which are being pursued in ongoing research. Becky Bisbee and Jeanne Darling have been invaluable resources for getting things done.

For helping me find a place to be a hermit, I thank Vernon and Beth Ingram and the people of Ashdown House.

I thank my family for years of support. My parents, Frank C. Sheldon and Beverly J. Sheldon, have always been a part of my education and have encouraged me all along. I acknowledge the support of my brother and sister, Eric and Pam, as well as of my Grandmother Evelyn Kist. My wife Ishrat Chaudhuri, deserves more than I can ever repay for her patience, good humor, and loving support.

Finally, I owe a great debt to the ballroom dance community for its support and for all the joy I have gotten from dancing. Special thanks are due my dance partner, Juliet McMains, for helping me during the most trying times. I thank my many coaches, especially Suzanne Hamby and Dan Radler. My previous dance partners have been absolutely essential to my mental health and happiness: Mercedes von Deck, Yanina Kisler, Yi Chen, and Elizabeth Earhart.

THIS THESIS IS DEDICATED TO MY PARENTS.

This thesis describes research done at the Laboratory for Computer Science at the Massachusetts Institute of Technology, supported by the Defense Advanced Research Projects Agency of the Department of Defense contract number DABT63-95-C-0005.

Contents

Chapter 1

Introduction

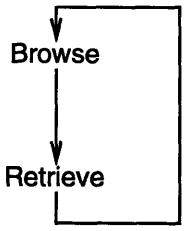
The Internet contains tens of thousands of information servers specializing in topics such as news, technical reports, biology, geography, and politics. The Internet's vast collection of servers can be viewed as a distributed database containing a wealth of information. Unfortunately, this information is relatively inaccessible because there are no satisfactory mechanisms for organizing the data or browsing and searching it effectively.

This thesis presents a new approach to the problem of information discovery and retrieval in very large networked environments. A hierarchical *content routing system* provides the user with the ability to do interleaved browsing and searching of a large, distributed information space at various levels of granularity. The goal of the design and implementation of the content routing system is to answer the following six questions:

1. Can a system help a user cope with the vast amount of information available by providing some organization of the data and automatically suggesting improvements to users' queries?

2. Is it possible to route queries to a meaningful subset of a large number of information servers?

3. Is it possible to define some metadata structures that can be used to organize the information space and provide other assistance to users?



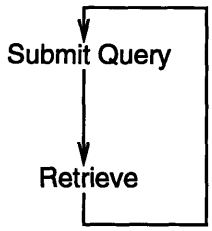


Figure 1-1 Browsing **Figure 1-2** Searching

4. Can the design successfully interoperate with pre-existing systems?

5. Is there a practical way to extract useful metadata descriptions of server contents from large numbers of disparate, incompatible servers?

6. Is it possible to build a system that scales to millions of servers?

The results of this thesis demonstrate that questions 1–5 may be answered affirmatively. The work thus far suggests that the content routing system may scale well, but a conclusive answer to question 6 will require more ambitious experiments.

A user's interactions with an information system span a broad spectrum between *searching* and *browsing*. At one end of the spectrum, the user knows exactly what document is desired and would like to *search* for it, perhaps by name. At the other end of the spectrum, the user has no concrete information need and merely wishes to *browse* to become familiar with the contents of the information system. In practice, a user starts with a vague or poorly specified information need that becomes more precise as she learns about the contents and organization of the collection. (This description of the spectrum between searching and browsing is a paraphrasing of an argument in [11].)

Thus there are three broad categories of operations of concern to a user of an information discovery system: document retrieval, browsing, and searching. The low-level operation of document retrieval in computer networks has been provided in many ways, notably by the file transfer protocol `ftp` [37] and the hypertext transport protocol `http` [4].

More recently, systems focusing on browsing have become available. Network file

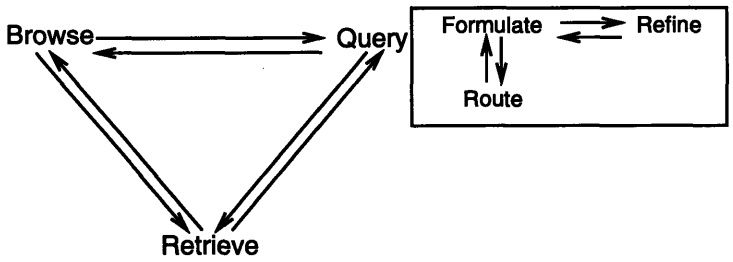


Figure 1-3 More complete model of user interaction

systems [66] allow users to browse file systems distributed across many machines on a network. Larger scale, indeed global, browsing systems such as gopher [1] and the World-Wide Web [5] are quite recent. A typical system for browsing provides the functionality of Figure 1-1. A user of such a system examines the contents of a document or a collection of documents given in a directory or a hyperdocument and gradually becomes familiar with the contents and structure of the available data. Truly global browsing systems can become tedious to explore, and it can be difficult to recall a path to an interesting documents even after it has been found. Existing browsers try to ameliorate this problem by allowing a user to save references to interesting places in the system.

Systems for searching the Internet are very new indeed, and nearly all of these systems have focused on the construction of centralized, global indices (*e.g.*, Archie [17] and the Web Crawler [47]). A typical system for searching provides the functionality of Figure 1-2. A user formulates a query that describes her information need. The query must be written in some query language, though plain text keywords are the norm. The system finds all documents on the network that match the query and presents the results, usually using some relevance ranking system. Search systems are difficult to use in the absence of a concrete information need or when naive queries lead to either too many or too few results.

Although some systems (*e.g.*, Gopher [1]) allow the user to browse until she finds a searchable index, systems have artificially partitioned themselves according to whether they provide a searching or browsing facility. However, because a user's information needs can change while using an information discovery system, it is important that a system

provide browsing and searching functions together. Figure 1-3 illustrates the complex interplay of browsing and search activities necessary for effective navigation of a large information space. This flexible model of user interaction is a principle contribution of this research. A user should always have the option to browse or search (using queries) as well as retrieve documents. These operations should be repeated at different levels of detail until the user's information need is satisfied. The query process itself should be interactive with the system suggesting refinements to user queries that will better match the user's interests and the system's performance capabilities. Suitably refined queries can be routed to appropriate remote information providers and the user may peruse the results using all the available tools. In the figure, a user may switch from any query operation back to browsing and retrieving (though for clarity the arrows have been omitted).

Progressive discovery is a model of interaction where the system provides a more detailed view of the information space as the user provides a more precise account of her information need. The system tries to provide enough detail to allow the user to proceed without overwhelming the user with a flood of information. Progressive discovery thus helps the user strike a better information bargain by providing more detailed information as the user supplies more information. The information supplied to the user is given at an appropriate level of granularity so that the user is not overwhelmed by a huge number of overly detailed responses.

Progressive discovery controls the complexity of a large information space, and for this reason it underlies much of the design presented in this thesis. No previous system provides an architecture that allows iterative, integrated use of browsing and search tools at various levels of granularity.

The *content routing system* employs a user-centered design that provides users both with the ability to browse multiple, coexisting information hierarchies and with associative access to the information. In particular, a content routing system combines interleaved browsing and search with two key services in support of associative access: *query*

refinement and *query routing*. Query refinement helps a user describe her interests more precisely by recommending related query terms. For example, if a user is interested in `buddhism`, the system might recommend query terms such as `zen`, `tibetan`, or theravada, which are all types of buddhism. Once a query has been refined and describes a manageable result set, query routing forwards the query to remote information servers and merges the results for presentation to the user. Query refinement and routing use information from compact descriptions of remote server contents called *content labels*. Experience with several implementations suggests that query refinement in conjunction with query routing us an effective tool for resource discovery in a very large document space. Moreover, we are convinced that query refinement is an *essential* component of any very large scale, rapidly changing information retrieval system.

This chapter will set the stage for the principled design of a very large scale, distributed information discovery and retrieval system. Previous work has focused on particular subproblems or has been driven by particular implementation technology. The approach here is user-centered. We first examine the problem from the user's point of view to determine what qualities are necessary for a usable system. Only then will we examine the requirements from the system's point of view. The content routing system design will be based on the principles that emerge from this analysis.

The remainder of this chapter

- details the problem of information discovery in large distributed systems and deduces necessary features of any system that is to solve the problem (Section 1.1)
- provides an overview of the content routing system design showing how it solves the listed problems (Section 1.2)
- describes related work explaining how previous systems have failed to meet important goals (Section 1.3)
- and summarizes the contributions of the thesis (Section 1.4).

1.1 The Problem

The difficulty of providing access to a large number of distributed information servers lies primarily in problems of scale, both for the user and for the system. Section 1.1.1 analyzes the user's needs first, and Section 1.1.2 examines the system's requirements. Each section will contain a set of necessary principles or features together with a brief argument for their necessity. These lists may not be exhaustive and the features are not orthogonal. However, I am unaware of any other comprehensive list in the literature, and there is no existing system that takes all of these principles into account.

1.1.1 The User's Perspective

The user's view of the information on the present day Internet is that of a vast space of documents clustered into groups in a large, flat, disorganized set of servers. Thus the user must cope with a vast, featureless expanse of information. To help the user negotiate this space, an information discovery and retrieval system should provide features to the landscape, allow the user to deal with a controlled amount of material at a time, and provide more detail as the user looks more closely. (This physical metaphor suggests a possible connection between data visualization and geographical information systems. Exploring this synergism remains for future work.) In fact, the system must be an active participant in the negotiation of an information bargain in which the user receives greater returns on greater investments.

The meta-principle of progressive discovery therefore underlies many of the principles in this section, and many of the principles are interrelated. The system helps the user learn more about the information space, which helps the user learn more about her information need, which helps her focus attention on particular parts of the information space, which allows the system to provide more detailed feedback. A critical corollary of progressive discovery is that the user must not be overwhelmed with more information than she can handle effectively.

The remainder of this section enumerates various features required by a user of an information discovery system for a large, distributed collection of information servers.

Single point of contact: The user needs a single point, or a few select points, of contact to begin. Even an expert user who has already found relevant sources needs a single point to return to in order to keep up with the additions and changes to the set of resources. In addition, a user requires mechanisms for exploring the information space even if she has only an imprecise idea of what information is sought. In fact, the user may not have any definable information need, just a desire to learn what is "out there."

Consistent interface: There must be a single, consistent user interface to the information discovery system. It is too difficult for the user to understand a wide variety of idiosyncratic interfaces. The Conit system [34, 35] has addressed this particular issue directly by providing a uniform, sophisticated, interface to a variety of existing information systems.

Content-based naming: In order to locate documents, a user requires a way to name documents according to their contents. Today, retrieving a document on the Internet requires that one know its precise location and its name at that location. The World-Wide Web has made document retrieval easier, but a document's URL (Universal Resource Locator) is still a location based name. These arbitrary names are impossible to guess and difficult or impossible to remember. Content-based names are names of documents that are derived from their contents. They are not necessarily query-based. For example, one could choose the Library of Congress subject catalog as a basis for naming documents. Every time a document is added to the collection it is assigned a path name based on its position in the subject classification and its title. Such a name is not based on the object's physical location. Given enough knowledge about the contents of an item and the classification scheme, one may deduce the item's name. (There are other reasons for separating naming from location. In fact, [63] argues for low-level names (Uniform Resource Names) that have no user-friendly semantics whatsoever. This thesis is concerned with higher-level names that users may guess or at least remember.)

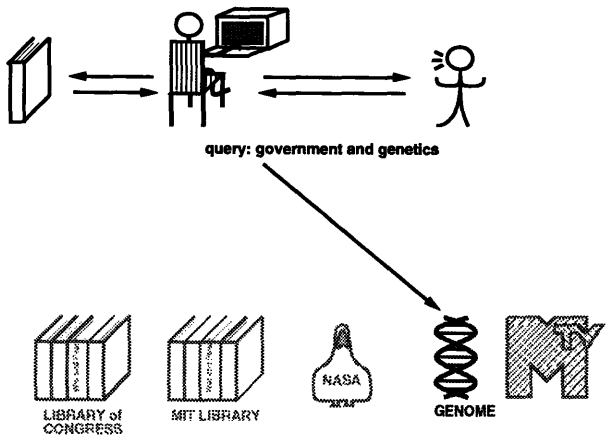


Figure 1-4 Seek off-line assistance

System-provided structure and feedback: The user must be able to rely on the information discovery system itself to provide the organization and tools necessary to find items of interest. First, users simply cannot use a large amount of information without some organization. Furthermore, because so many information servers and so many documents already exist or are being created, no one person can know what is available on the Internet on an arbitrary topic. The rapid change in the Internet's contents also implies that any guidebook is out of date before it reaches the bookstores. Figure 1-4 shows a user contacting a particular information server after consulting published references and colleagues. Thus, while off-line resources such as human experts and reference books will always have some utility, the system itself is the only authoritative source of information about the system's contents, and it must be able to provide the necessary help to the user.

Browsing: Again, a user without a readily specifiable information need requires a mechanism for discovering the system's contents. It must be possible for the user to browse the system's contents as one would a library. However, the system must not overwhelm the user with a barrage of information. Providing a service that lists all documents and/or all hosts on the Internet, for example, is simply not useful. In order to support progressive discovery while browsing, there must be some organization of the

information space, perhaps based on a subject classification system.

Multiple, over-lapping hierarchies: A large scale information system ought to provide browsing of multiple coexisting, even overlapping hierarchies, each of which presents a different view of the information system's contents. Hierarchical structures organize an information space so that a user can browse it effectively using progressive discovery. Libraries have used hierarchical classifications since antiquity and continue to do so today. Subject-based hierarchies like the Library of Congress cataloging system are clearly useful tools for organizing information. Different types of subject hierarchies exist and have proven useful, even rather *ad hoc* schemes such as the WWW Virtual Library.[1] In addition, users may find other types of organization useful. For example, a geographically-based hierarchy may help one who knows that relevant work was done at some particular university. Similarly, institutional hierarchies can be useful. (Organizations of hierarchies are discussed in more detail in Section 2.3.4.) Since users may have different information needs or different ways of describing their information needs, there should be multiple hierarchies providing different views of the data.

Associative access: Users require associative access to both documents and internal clusters in a hierarchy. Any hierarchical structure, however well organized, can become tedious and inefficient to search, especially as a user's needs become more concrete. Thus, users with more specific information needs require a more direct path to items of interest. However, the principle of progressive discovery warns against simply providing direct associative (or query-based) access to all documents in a large system. Using the hierarchical organization of the information to group documents into clusters can allow one to gauge the size and relevance of the result set of a preliminary query (see below).

Query refinement: Users require some automatic means to focus queries more precisely. Because the information space is so large and rapidly changing, initial user queries will have huge result sets that will overwhelm the user. It is not in the user's interest (or in the interests of the discovery system) to process such queries. The system

[1] `http://info.cern.ch/hypertext/DataSources/bySubject/Overview.html`

must reveal high processing costs to the user before they are incurred, but it is not enough just to inform the user that a query is too expensive. The system must actively help the user formulate better specifications of her interests. One way to do this is to allow a query-based form of progressive discovery in which the user finds out about large segments of the hierarchy that are relevant rather than individual documents. But even this may yield unmanageable results on large servers. It is therefore incumbent on the system to help the user narrow the search space by suggesting query modifications that will be more precise and efficient. The system must do this because it is the only source of expertise available to the user. Suggestions for improving a query must be related to the user's interest and also to some on-line knowledge of the network's contents. *Query refinement* is a facility that, given a user's query, automatically suggests query terms that will help the user formulate a better query, one that more accurately reflects the user's interest and can be answered within practical performance constraints. Figure 1-5 shows one of the prototype content routing systems refining the query `buddhism`. This query has identified a variety of document collections too numerous to browse (the list goes off the bottom of the window). The system's suggested terms are in the pop-up window, and the user may click on any term to add it to the query. The algorithm that computes the list is very simple and is described in Section 2.3.2. Many of the suggested terms have a strong semantic relationship. For example `tibetan`, `theravada`, and `zen` are all types of Buddhism. Other terms are more related to service type, for example `ftp`.

Query Expansion: Often however, queries produce few or no matches, perhaps because the query terms are not indexed at all. In such cases, it is useful to find related terms to add to a query to *increase* the result set size. There are a variety of standard techniques for query expansion. Thesauri and term collocation statistics have been used to increase the size of result sets to improve recall. See [51, pp. 75–84] as well as [64, 48, 15]. This is the opposite of the purpose of query refinement. Section 2.3.2 discusses how to parameterize the query refinement techniques used in the prototype content routers to serve for both query refinement and query expansion.

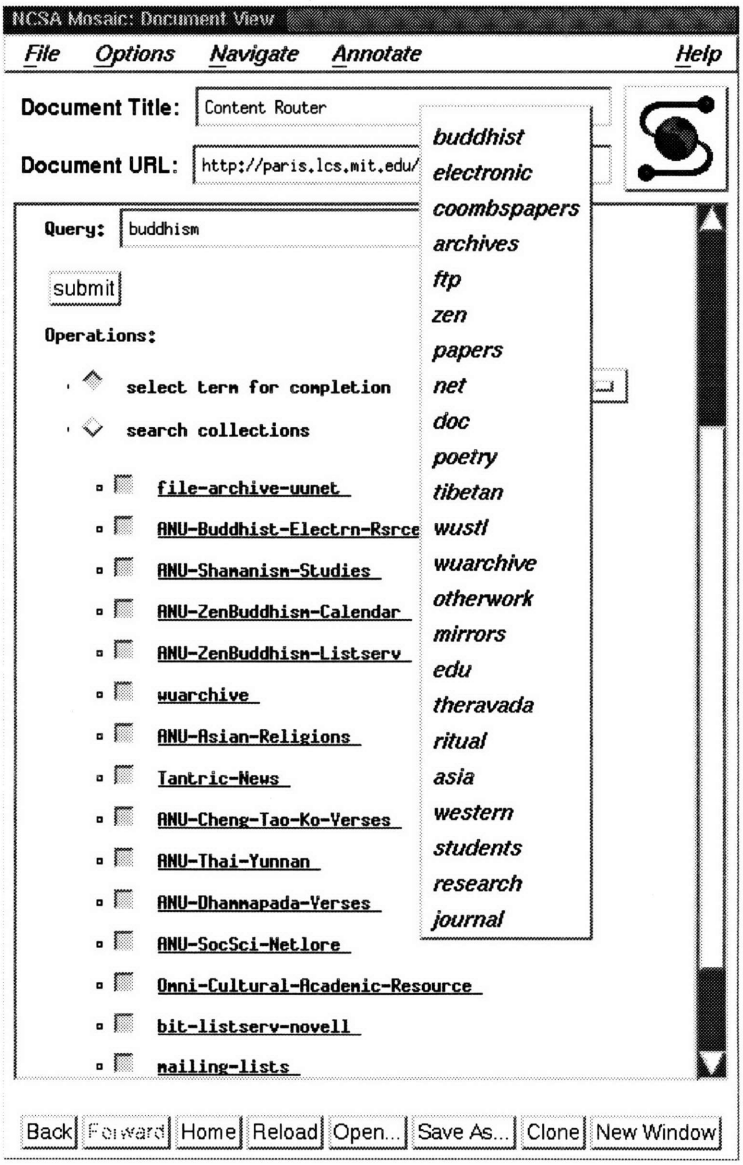


Figure 1-5 Example of query refinement.

In summary, any very large information discovery system must provide a way to deal with the complexity of the search space both for browsing *and* for querying. The system must provide feedback *and* suggestions on partial or poorly specified queries if it is to be successful in a large, heterogeneous, dynamically changing environment such as the Internet.

1.1.2 The System's Perspective

This section reviews the technical challenges a general, large scale, distributed information system faces in modern networks. Since the system's perspective has been the focus of prior work, this review is brief. Note that the system requirements are not necessarily the same as the user's requirements. It may be that user requirements are impossible to meet with present technology (or at all). Happily, this is not the case.

Scale: Any large, distributed information system must be concerned with scale. From the system's point of view, scalability has to do with the management of resources. A scalable system is one that can grow without resource requirements anywhere in the system exceeding the resources available. In general, as the number of documents gets larger, there must be no point in the discovery system where resources grow too quickly.

Performance: Any system must provide acceptable performance to be useful. Performance, however, is not a monolithic measure. Response time is, of course, important, as are space requirements. Equally important is the quality of the results. The system must do an acceptable job of helping users find the information they seek. Traditional measures of *recall* (finding as many relevant items as possible) and *precision* (not returning irrelevant items) are critical. Ultimately, in a large, distributed environment, an implementation will trade off these various measures of performance.

Network usage: This item is related to both scale and performance. The information system must not flood the network with traffic that reduces the bandwidth available for other work.

Heterogeneity: The Internet today has a very heterogeneous collection of information servers that is changing rapidly. Servers can be personal computers, large file servers, or even powerful multiprocessors. This, together with the distributed nature of the information system, suggests that it is important to for the implementation to tolerate varying latencies and capabilities in its operations.

Interoperability: A corollary of heterogeneity is interoperability. If a wide variety of system types are to be included in a large distributed system, then they must interoperate. There is a large investment in different types of information systems on the Internet, and it is essential to interoperate with these systems to leverage their large information resources.

Autonomy: Again, wide participation suggests that many systems will be unwilling or unable to comply with onerous restrictions on data organization and even semantics. While it may be possible to organize cooperating sets of information providers that abide by a fixed set of rules (for example, naming systems do this), it is important that a general information system design allow for more flexibility.

Ease of participation: The interactions with various servers need to be very simple to support a wide variety of systems. Also, the information system needs to have a very dynamic structure. For example, it must be relatively simple for a new server to join the information hierarchy (or to have data enter the system). A new server must be able to request registration with the system, and the system must be able to request that a server register. A simple interface and simple registration procedures are essential to encourage participation.

1.2 Content Routing System Overview

The content routing system design described in this thesis incorporates the features of Section 1.1. The detailed design of the system will appear in Chapter 2, which includes a detailed example session in Section 2.1; however, this section provides a quick overview.

The content routing system supplies a single logical point of contact, which is the root of a browsable and searchable information hierarchy. The interface is a simple extension of both a distributed file system and a conventional information retrieval system. We have previously explored the combination of file and information retrieval systems [21], though this design is not limited to that particular file system model. Interaction with the system allows browsing, as in a distributed file or hypertext system. A content routing system extends the combined file and information retrieval systems with the notion of a *collection* document, which is a cluster of related documents. Browsing and searching involve the use of collection documents. As the user's attention focuses on a manageable number of collections, the collections may be expanded or searched to expose more detail about their contents and to allow more precise queries. This allows the user to see a controlled amount of information, and, as the information need becomes more precisely specified, the view of the information space becomes more detailed. Thus, the clustering of documents into collections supports the exploration of the information space at varying granularities.

Collections are supported by the idea of a *content label*, which is a compact description of the contents of a collection (*i.e.*, the contents of an information server). Users may look at content labels to help in the formulation of queries as well as to understand the contents of a collection. Content labels are used for routing queries to relevant servers and for providing query refinement suggestions. Content labels are chosen by the systems that administer the collection, that is, they are determined at the site where the information is. This thesis explores some techniques for automatic construction of content labels.

Names of items within a content routing system depend on the hierarchy, not on the item's physical location. In particular, content routing systems support subject-based hierarchies for content-based naming, and they support search tools for associative access. The information hierarchies structure the information space, and the query refinement feature provides feedback tailored to the user's needs and to the user's interests and current position in the information hierarchy.

The hierarchical structure provides a classic solution to one aspect of the problems of scale. A content routing system consists of a directed graph of information providers, each representing a collection. The internal nodes are *content routers*, which are servers that implement the design of this thesis. These content routers will have information specialties according to the hierarchy they are in. Whenever the number of elements at a node gets too large, some elements are gathered together into a new node. An information discovery system may exploit the semantic hierarchy demanded by the users' needs as outlined above. The physical hierarchy need not, of course, reflect the semantic hierarchy exactly.

A second feature that supports scaling is the use of content labels. While a hierarchical structure controls fan out, content labels provide a way of locally controlling the required indexing resources. For example, a content router that has ample resources and indexes small collections may allow arbitrarily large content labels. However, a content router may also require content labels to be of a particular size, or may charge for extra content label space.

Naive strategies of query processing also raise the issue of scale. Figure 1-6 shows a system that simply forwards a user's query to every server in the system. Though such a system provides a single point of contact for the user, it wastes enormous amounts of network resources and suffers poor performance as the numbers of users and servers grow. While various strategies of replication and caching may ameliorate these problems in the near term, a global broadcast scheme will not scale because it is too costly and inefficient, enormously complicated, or both. Moreover, it avoids the central issue of how to help the user understand the information space.

Figure 1-7 shows another naive approach to the problem. Here, the system builds a global index of all documents and makes it available to the user. This is the approach taken by systems like Veronica [26], Lycos [36], and the RBSE Spider [16]. The scale of the Internet, which is today only a fraction of its eventual size, is so great as to render infeasible any comprehensive indexing plan based on a single global index. Even though

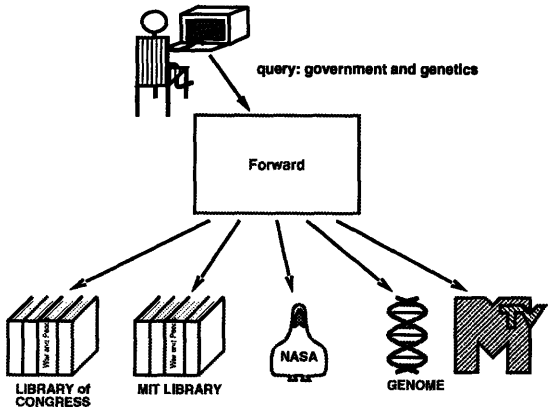


Figure 1-6 Broadcast query everywhere

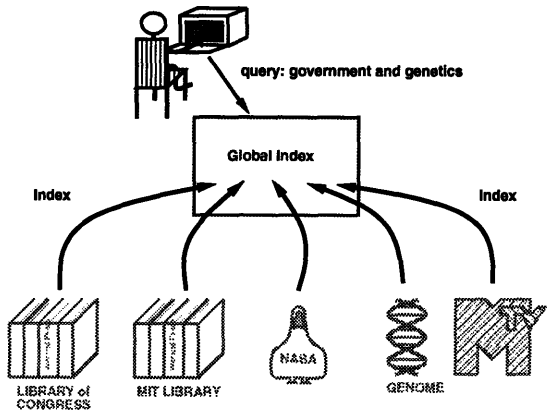


Figure 1-7 Use a global index

some sites on the Internet distance themselves from the rest of the network via firewalls and other techniques, the trend is toward greater and greater connectivity [56]. A global, centralized index will grow with the size of the document space and will be very difficult to keep up to date. Though its performance for the user is much better than that of the broadcast approach, it ultimately will not scale. It already takes Lycos months to scan the Web from the documents it stores as starting points for a search. The Canadian company Open Text is preparing to offer full text indexing of the entire Web, and even at its current modest size, they expect a database of 40–50 gigabytes managed by three dedicated DEC Alphas [41]. Transmitting all documents on the net to the index server for indexing (or even the index data which is still large) will represent an enormous amount of network traffic. This traffic, if the global index is to be kept up to date, will reduce available network bandwidth considerably and create local bottlenecks. Organized use of a wide distribution of indexers and managed communications procedures ameliorate these problems, and in fact, this is one way to view the content routing approach. Furthermore, a global indexing strategy does nothing to organize the information space for the user.

The architecture proposed in this thesis is a compromise between broadcast and global indexing. Figure 1-8 shows a content routing system accepting a user query and forwarding it only to relevant servers. To do this, the system must have some idea of server contents. A content routing system uses a compact description of server contents

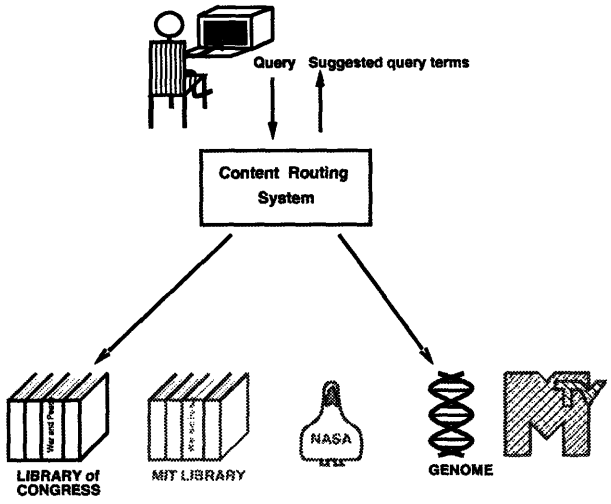


Figure 1-8 Route only to relevant servers

called a *content label* to perform query routing and refinement. A content label may be small compared to the size of an entire collection, unlike the information required for a full global text index. Furthermore, as Chapter 2 will show, content labels can be designed so that they change more slowly than the collection itself. A hierarchical arrangement of *content routers* can also be thought of as containing a distributed global index with index data becoming more complete as one approaches the leaves of the network. The architecture supports a flexible registration protocol and allows a great deal of server autonomy.

Thus, this thesis explores the design of a content routing system that:

- Supports browsing *and* associative access.
- Supports a uniform query interface that allows progressive discovery of network contents and guides users in formulating queries of adequate discriminatory power.
- Propagates descriptions of server contents through a hierarchy of information servers.
- Routes individual queries to available servers based on the expected relevance of the server to the query.

This thesis will *not* concern itself with certain issues that are clearly important for a full scale system. Namely, I will not address replication, network protocols or naming (accept for the specific navigation and query tools in the design). I will also not address the important issue of deduplication, that is how to remove duplicate documents from the system. Duplicates can arise from the same document's presence in more than one server, different versions of the document being available, or by different access paths to the same document. The last possibility can easily be overcome if there is a system for normalizing names [28]. See Section 4.1 for a brief discussion of these and other issues for future research.

1.3 Related Work

The content routing system design is unique in its use of query refinement. Perhaps this service has been neglected, while query expansion has long been available, because of the pervasive use of the vector space query model. (See [51] for information on vector models). Since vector models do not support boolean conjunction, there is no way to reduce a result set by adding query terms. New query terms always increase recall. Our use of a boolean query model makes query refinement a practical feature rather than just an ideal. The inclusion of the query refinement feature draws on our past experience with the Community Information System [22, 20]. In the Community Information System, a simple theorem prover assisted the user in choosing query terms that guaranteed the query could be satisfied at available news wire databases.

Related work can be broken down into the following categories: Global indices, subject directories, network-based information retrieval systems, and network-based resource discovery systems.

1.3.1 Global Indices

Web robots like the Web Crawler [47], ALIWEB [30], Lycos [36], the RBSE Spider [16], and Yahoo[2] gather information about resources on the Web for query-based access. Veronica [26], a discovery system that maintains an index of document titles from Gopher [1] menus, uses the same strategies. The Archie system [17] polls a fixed set of FTP sites on the Internet and indexes the file names on those sites. A query yields a set of host/filename pairs which is then used to retrieve the relevant files manually.

All these systems use a global indexing strategy, *i.e.*, they attempt to build one database that indexes everything. Even so, they limit the information they index: for example, Lycos indexes only the title, headings and subheadings, 100 words with highest weights (according to a weighting function), and the first 20 lines. They further restrict the size of the indexed data in bytes and in number of words. If there must be restrictions on indexed terms, it would be preferable to leave the decisions to entities that have domain specific knowledge (as well as knowledge about the users of the data). Global indexing systems do not provide any organization of the search space, and they do not give the user any guidance in query formulation. These systems overburden network resources by transmitting all documents in their entirety to a central site (or a cluster of indexing sites in the case of Lycos). The futility of this sort of architecture is already apparent: it takes on the order of weeks (or more) for Lycos to update its indices. A content routing system allows distributed processing of information at more local intermediate servers, thus distributing the work load and avoiding network bottlenecks. Furthermore, a content routing system allows greater autonomy to each information provider and content router to tailor its indexing mechanisms and facilities using local knowledge.

One interesting feature of Lycos is its use of pruning the graph of Web documents it has indexed (in order to preserve a finite representation). When the graph is pruned, a place holder containing the 100 most important words in the documents that were pruned is left behind. This list is built from the 100 most important words in each of the

[2] `http://www.yahoo.com/docs/info/faq.html`

pruned documents. This can be viewed as a very ad hoc and rudimentary type of content label. Another interesting feature of Lycos is its association of link information (the text describing a link) with the document the link refers to: some information from the parent is indexed to refer to the child. This information is useful because it is essentially a name or description of the document referred to thought up by someone else. This information is not necessarily available at the site where the document resides, and thus may be difficult to integrate into a content routing system.

Yahoo also is different from the other systems in this category because, like the content routing system and the Semantic File System [21], it provides a hierarchical organization of the data for browsing. In fact, Yahoo can be viewed as a Semantic File System implemented in the World-Wide Web. The hierarchical organization is ad hoc, but quite useful. Queries are processed against the entire document space as if the hierarchy were flattened. One very useful feature (also true of the Semantic File System) is that query results are enumerated with their entire path name in the hierarchy. This allows the user to submit queries and use the results to learn about the hierarchy, possibly following up previously unknown clusters of data.

1.3.2 Subject-Based Directories

Subject-based directories of information, *e.g.*, Planet Earth[3] and the NCSA Meta-Index[4], provide a useful browsable organization of information. These systems focus only on browsing. These hierarchies would be useful paradigms for organizing particular hierarchies in a Content Routing System. However, as the information content grows, a browsing-only system becomes cumbersome to use for discovering relevant information. Associative access is necessary for quickly generating customized views of the information space in accordance with a user's interests, and this associative access must work at varying levels of granularity. None of these systems provide this integration, nor do they

[3] `http://white.nosc.mil/info.html`
[4] `http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/MetaIndex.html`

provide query routing and refinement.

In contrast to most of these systems which use static, externally-imposed hierarchies, the Scatter/Gather work of [11] uses data-driven, automatic construction of information hierarchies based on fast clustering algorithms. I expect that many useful content routing hierarchies could be developed using their off-line clustering algorithms, and it may even be feasible to blend the Scatter/Gather dynamic clustering techniques with content routing (though this remains an open question). Another approach to data-driven hierarchy construction may lie in clustering by automatic theme extraction [53]. If the past is any guide, automatic clustering may ultimately prove more useful than manual clustering just as automatic indexing has proven to be preferable to manual indexing [52]. Nonetheless, content routing supports hierarchies based on data-driven clustering as well as on other organizing principles.

The Scatter/Gather work does take a similar point of view to that of content routing, namely that the user model should allow for progressive discovery with interleaving of browsing and searching at varying levels of granularity. In [49], the authors use the term 'iterative query refinement' to mean what I call 'progressive discovery.' The Scatter/Gather architecture does not seem to support distributed processing where clusters are implemented as servers, and there is no notion of query routing or refinement. Users must explicitly select clusters for further interrogation.

1.3.3 Network-Based Information Retrieval Systems

Network-based information retrieval systems provide associative access to information on remote servers. WAIS [29] combines full text indexing of documents with network-based access. However, there is no facility for routing queries to relevant databases and merging results, nor is there any mechanism for suggesting query terms to users.

The Conit system [34, 35] provides a uniform user interface to a large number of databases accessible from several retrieval systems. User queries are translated into commands on the different systems. However, there is no support for the automatic

selection of relevant databases for a user's query.

1.3.4 Network-Based Resource Discovery Systems

Network-based resource discovery systems like Harvest and GlOSS gather information about other servers and provide some form of query-based mechanism for users to find out about servers relevant to a request. Harvest [8] builds on our work on content routing by providing a modular system for gathering information from servers and providing query-based browsing of descriptions of those servers. A broker is a kind of content router, and a SOIF (Structured Object Interchange Format) object is a kind of simple content label. However, Harvest has no architecture for composing brokers. The Harvest Server Registry is like the WAIS directory of servers: It does not appear to support interaction with multiple servers by query routing and merging of result sets; rather it supports the ability to click on a broker and query it. There is no query refinement capability or even enumeration of field names or values. The user must browse through the SOIF objects to get hints on how to construct queries.

The GlOSS system [25] also provides a mechanism for finding servers relevant to a query, but it uses a probabilistic scheme. GlOSS characterizes the contents of an information server by extracting a histogram of its words occurrences. The histograms are used for estimating the result size of each query and are used to choose appropriate information servers for searching. GlOSS compares several different algorithms using the histogram data. The content routing system prototypes do not fit into any of GlOSS's categories, because they do not use document frequency as a measure of server relevance for routing (though they do use document frequency data for suggesting query refinements). Moreover, GlOSS's estimates of query result set sizes are not at all accurate. These estimates, which are used for a kind of routing function, are inaccurate because GlOSS assumes query terms appear in documents independently of each other and with a uniform distribution. This assumption is obviously false, and all data-driven clustering algorithms as well the query refinement feature in a content router are based on the

certainty that term occurrences are *dependent*. It would not be such a problem for other aspects of cost estimation (or even parts of a query refinement algorithm) to be based on this false heuristic if it were to prove useful, because these other features do not affect the semantics of query routing and identification of relevant sources. GlOSS also does not search remote databases, it only suggests them to the user who must search them and retrieve documents manually. More recently, the GlOSS work has taken up the idea of hierarchical networks of servers, though this effort is preliminary and has been used only for two-level hierarchies [24].

Simpson and Alonso [61] look at the problem of searching networks of autonomous, heterogeneous servers. The focus of this work is on determining the properties of a protocol that preserves server autonomy while providing useful services. In particular, they decompose the problem into various independent modules and describe the operations each module implements. Their system does not provide any analog of query refinement, or even automatic query routing: the user has to select servers to search manually. It is also not clear whether any form of browsing is allowed. However, they did have an analog of content labels to describe server contents. Their approach has a probabilistic query semantics and does not support browsing with query refinement. Also see [2] for a survey of approaches to integrating autonomous systems into common discovery systems.

The Distributed Indexing mechanism [13, 12] is based on precomputed indices of databases that summarize the holdings on particular topics of other databases. The architecture has a fixed three layer structure.

The Information Brokers of the MITL Gold project [3] help users find relevant information servers, but they do not provide access to objects directly. Rather, they return descriptions of an object's location and the method that may be used to retrieve the object.

1.3.5 Distributed Naming Systems

Distributed naming systems such as X.500 [9], Profile [46], the Networked Resource Discovery Project [55], and Nomenclator [42] provide attribute-based access to a wide variety of objects. The content routing architecture, as will be described in Section 2.3.3, allows the use of Nomenclator style hierarchies as a special case. However, naming systems traditionally operate in highly constrained environments where objects have few attributes. In Nomenclator, object metadata consists of *templates*. A simple subset relation (together with the use of wildcards for some attributes) allow a server to identify relevant information servers and forward queries. Nomenclator restricts the semantics of the system so that a server must be authoritative for any query that matches its template. This represents a strong restriction on the structure of the information system. Moreover, it is clearly assumed that templates are relatively small and that a relatively small set of attributes can be found that split the search space and will be known to the user. Section 3.1 discusses the search for such terms in a particular set of servers. Also, [42] suggests that an *advise* operation, similar to a content router's query refinement operation, can help users specialize queries by listing all attributes in revised templates generated by remote executions of the query. In any general information retrieval application on the scale envisioned in this thesis, this would be impractical. However, the architecture does not allow structures that will support term rankings in the advise operation.

1.4 Contributions of the Thesis

The content routing system design represents a new approach to information discovery and retrieval in large distributed systems that:

- examines the interplay of offline data organization and online interleaving of browsing and searching at various levels of granularity.
- supports distributed processing and query routing for better performance and ease of integration.

- exploits heterogeneity of data sources to aid navigation.
- provides a query refinement feature in which the system automatically recommends query terms chosen based on the users query and knowledge of the system's contents.
- directly addresses the metadata issue by experimenting with automatically generated content labels that describe the information exported by a server.

Furthermore, this thesis work has examined some of the pragmatic questions raised by such a design through a series of prototype implementations. The implementations have demonstrated the feasibility of the content routing architecture by demonstrating the following research results:

- It is possible to give meaningful and useful suggestions to help a user formulate better queries.
- It is possible to do automatic query routing to a meaningful subset of a large number of information servers.
- It is possible to define metadata structures that can be used to organize the information space and provide assistance in formulating queries.
- The content routing architecture can successfully interoperate with pre-existing systems.
- It can be practical to extract useful content labels from hundreds of disparate, incompatible servers.
- A hierarchical content routing system scales well to hundreds or thousands of servers. I am guardedly optimistic that a well-organized system could cover millions of servers.

1.5 Structure of the Thesis

The remainder of this thesis is organized as follows:

- Chapter 2 gives the design of the content routing system architecture.
- Chapter 3 describes implementation efforts and experience with the resulting prototypes.
- Chapter 4 summarizes the results of the design and implementation efforts and describes directions for current and future research.

Chapter 2

Design of a Content Routing System

The goal of the content routing system's design is meet the user's needs as outlined in Section 1.1.1. Any system is constrained by the system requirements outlined in Section 1.1.2. The design of the content routing system exploits the substantial commonalities and compatibilities between the user's needs and the system's requirements. For example, a hierarchical organization of data supports browsing by structuring the information space. A hierarchical organization of data also provides the system with a strategy for decomposing the information space onto distributed servers. Similarly, the user's requirement for more detail as her information needs become more specific suggests that the system need not store or process detailed information when dealing with non-specific information requests. Requests for detailed information may be processed at specialized information servers that cover a narrower range of material more thoroughly.

A content routing system, therefore, consists of a hierarchical network of information servers that supports browsing and searching. To support associative access to its contents, a content routing system directs user requests to appropriate servers (query routing) and helps a user formulate meaningful queries (query refinement). Both of these tasks require a content routing system to employ detailed knowledge about what in-

formation is available and where it is located. This knowledge is encoded in metadata descriptions of the contents of remote servers called *content labels.*

This chapter presents the semantics and architecture of content routing systems:

- Section 2.1 contains an example session with a prototype content routing system.
- Section 2.2 gives the semantics of a content routing system by describing the abstractions and operations provided to the user. The section also details the structure of queries.
- Section 2.3 gives the architecture of a content routing system. It
  - describes how architectural components of the system match up with the semantic constructs.
  - outlines detailed strategies for query routing and refinement.
  - explores alternatives for constructing content labels and content routing hierarchies.

2.1 Example Session

Even though a content routing system uses many familiar concepts, it is best to illustrate the design with a running example. This will make the subsequent discussion of the abstractions and operations provided by the system more concrete.

A *content router* is an information server that supports the content routing system interface to provide access to other information resources. A content router registers a number of information providers and organizes them into a local directory structure, which may be browsed like a tree-structured file system or searched like an information retrieval system.

Figure 2-1 shows the initial contact with a prototype content router running on a particular information hierarchy. The prototype uses the notion of a *current result set.*

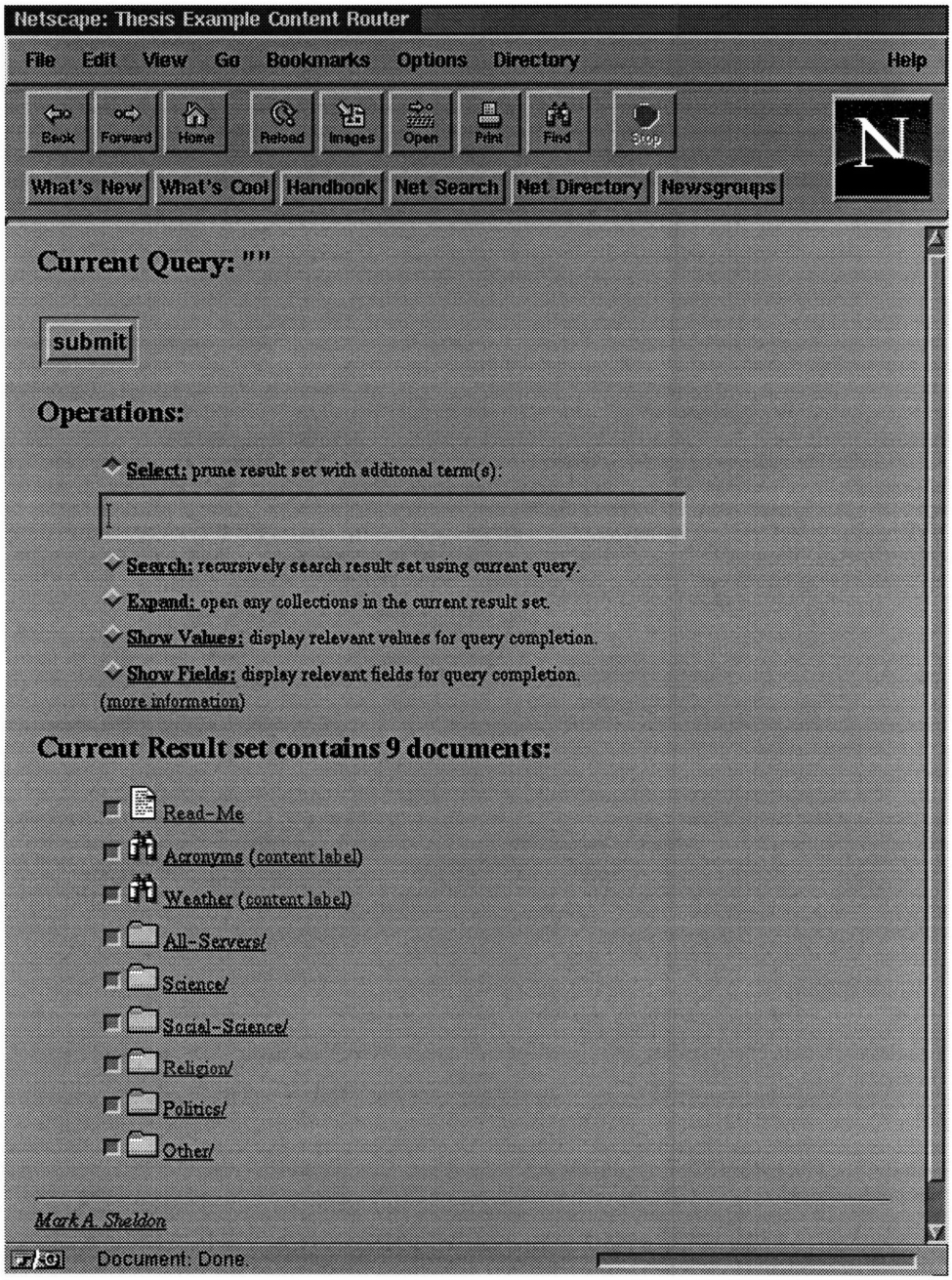


Figure 2-1 Top-level view.

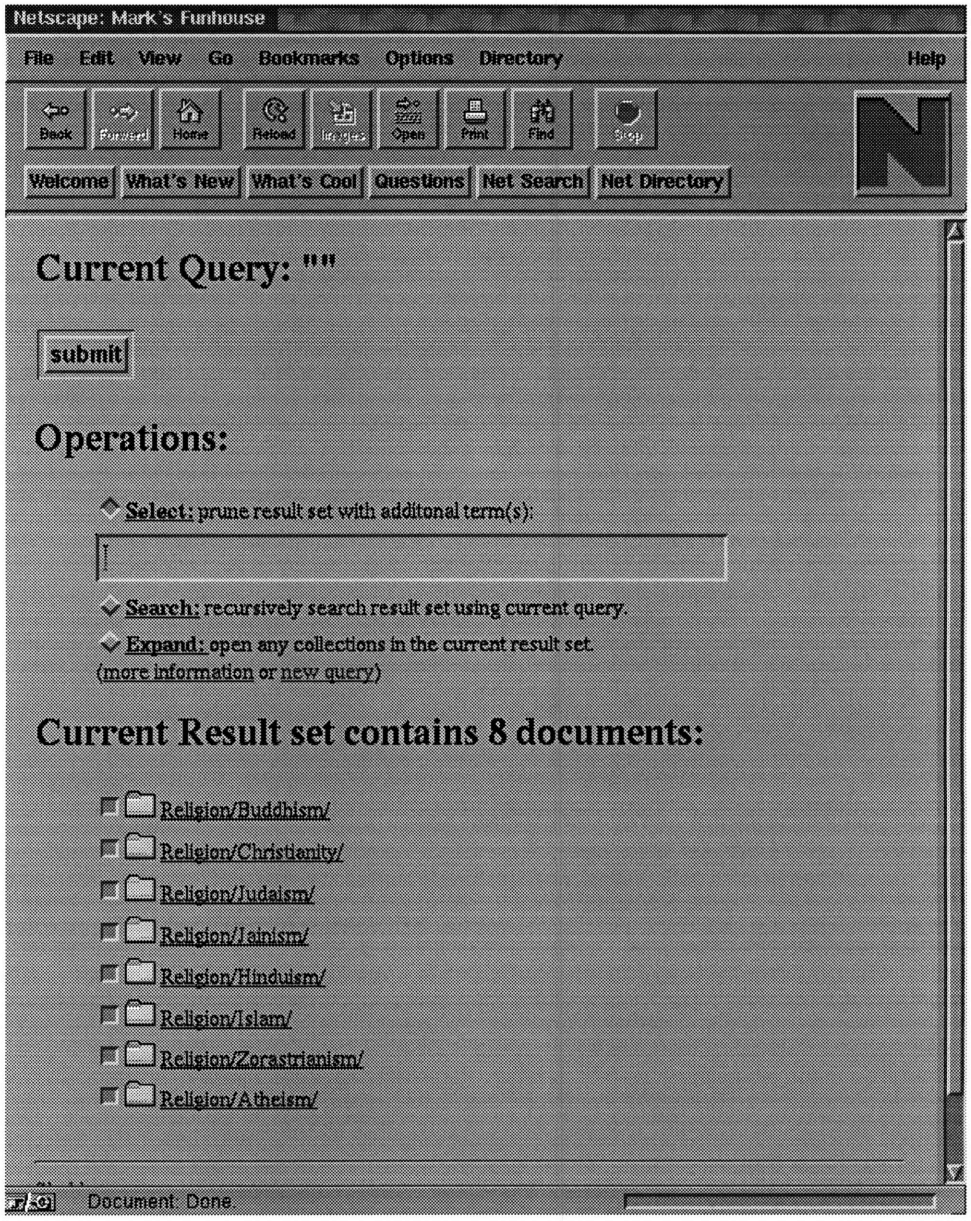


Figure 2-2 Expanding a directory.

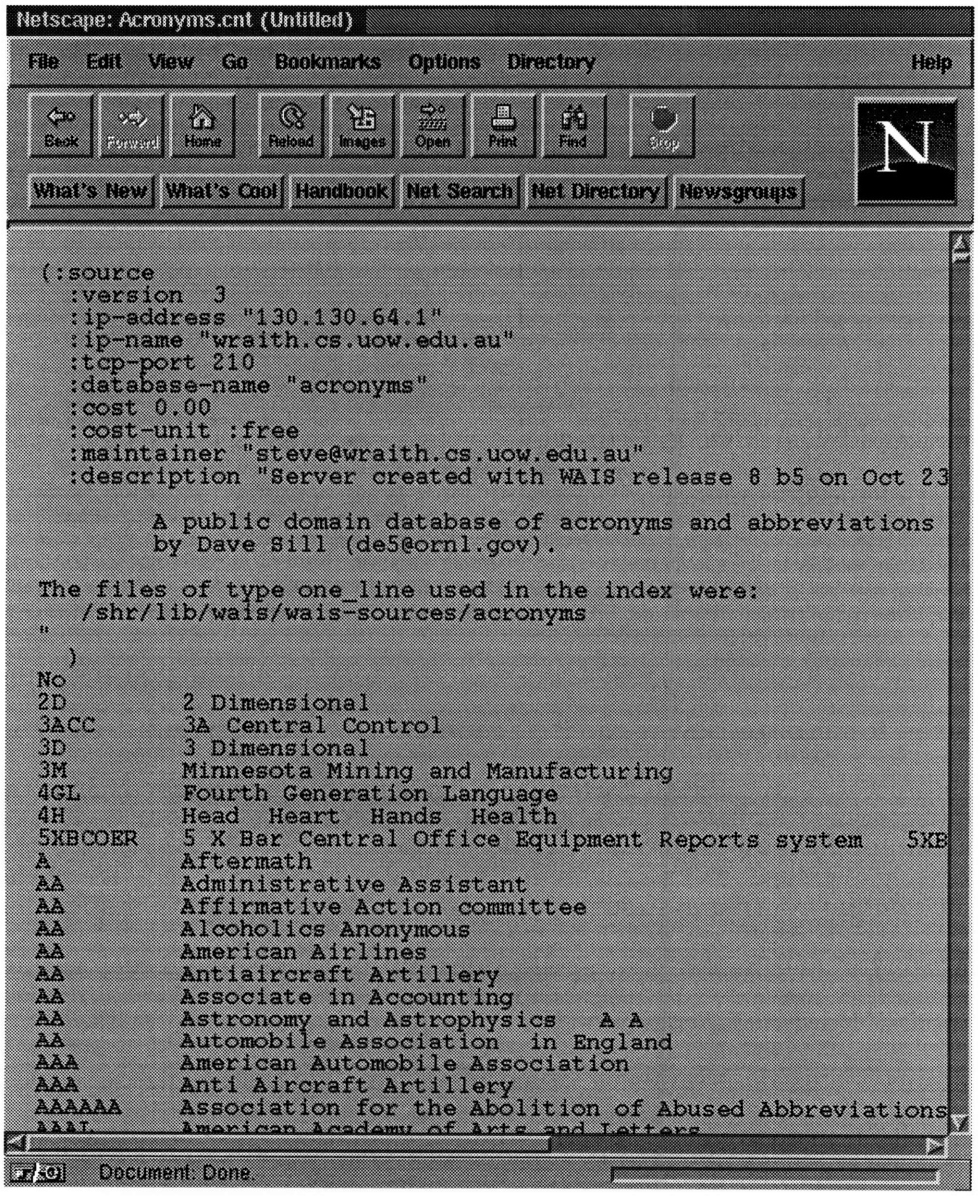


Figure 2-3 A sample content label.

The documents in the current result set form the roots of a hierarchical information space that the user may browse and search. This information space forms a *universe* for the various operations described below. The user may selectively remove items from the current result set by un-checking the boxes next to those items. The current result set is displayed at the bottom of the window.

The top-level of this content router contains nine documents in the initial result set. These documents fall into three types: base documents, directories, and collections.

Base documents can be any ordinary document that one might encounter in an information retrieval system: ASCII text files, PostScript documents, graphics, sound, and video are typical examples. In Figure 2-1, `Read-Me` is a plain text file. Base documents are retrieved by clicking on their names.

Most of the documents in the result set are *directories,* which in turn contain other documents. To support browsing, a content router uses directories to provide local structure to the information space, just as one uses directories in an ordinary file system or gopher. A user retrieves the contents of a directory by clicking on its name. Figure 2-2 shows the result of retrieving the `Religion` directory of Figure 2-1. It's components are all subdirectories devoted to particular religions. Browsing may continue in this manner just as in any file system browser.

The content routing system introduces the idea of a *collection document.* In Figure 2-1, `Acronyms` and `Weather` are collections. Collections, like directories, contain other documents. However, a collection also consists of a description of the set of documents it contains. This description is called a content label. Users may click on the name of the collection to connect directly to an information server for that collection, which may be another content router or a server that supports only base documents (and perhaps directories). Users click on `(content label)` to retrieve the content label for the collection. There are many choices for content labels. The design issues will be outlined below in Section 2.3.3, and Chapter 3 will discuss the decisions made for the prototype content routing systems. Figure 2-3 shows the content label on this prototype

for the `Acronyms` collection. Human readable content labels are useful for learning more about the contents of collections and also about available query terms.

The search operations appear under the heading `Operations`. These operations are used to build queries. The system displays the *current query* at the top of the window. This query is empty at initial contact and is unaffected by browsing, as can be seen in Figure 2-2.

Query-based search operations in the present content routing system design distinguish three cases that differ in the treatment of collections, and therefore, in the granularity of the search. Only two cases will be shown here. The complete set of search operations will be described in Section 2.2.3.

The `select` operation is finds all documents in the universe that match the query augmented by the new terms typed in by the user. For a `select`, and for all the search operations of the content routing system, the query is applied to base documents, directories, and collections in the result set, as well as to the contents of directories (recursively). The search is confined, however, to the local server without looking inside collections. Collections are treated as single documents that match a query if their content labels match the query. Figure 2-4 shows the result after the user types in the query term `tibetan` and submits the form in Figure 2-2 with the `select` operation. There are 13 documents, all collections, within the `Religion` directory that match this query. Notice that the local directory structure disappears as a result of a `select` operation, *i.e.*, the directories beneath the current result set are flattened for the selection of relevant documents.

If a search of all 13 remote servers seems too expensive, the user may wish to further restrict the size of the universe. If the user cannot think of any appropriate query modifications to accomplish this task, she may use the *query refinement* feature. The system responds to a query refinement request with a list of suggested terms that might be added to the query to narrow the search space. Figure 2-5 shows the terms suggested by the system for the query `tibetan`. (This figure comes from an earlier implementation

of the content router because, as of this writing, query refinement is still being reimplemented for the new prototype.) Many of the terms are very relevant to the query: `asian`, `tibet`, `buddhist`, `buddhism`, `meditation` relate to Tibet and its Buddhist traditions; `coombspapers` is the name of a collection of bibliographic materials on Buddhism; a `lama` is a Tibetan Buddhist priest, `oracles` are an unusual feature of Tibetan Buddhism; `sino` relates to sino-tibetan, the language family to which Tibetan belongs as well as a set of political relations; `yoga` is widely practiced in Buddhism; `thangkas` are religious paintings on scrolls; `tantra` relates to the Tibetan practice of tantrism (in Buddhism); and `saky`[a] is a school of Tibetan Buddhism. The presence of `ftp` derives from the existence of several ftp archives on the subject.

Figure 2-6 shows that the addition of the text term `meditation` has reduced the number of relevant servers to three. An exhaustive search of these servers is done when the user chooses the `search` operation. A part of the list of documents from all three collections is visible in Figure 2-7.

2.2 Content Routing System Semantics

2.2.1 Abstractions

An important challenge in the design of a system for information discovery in very large environments is to find a small and simple set of abstractions and operations that users can understand and use effectively. These abstractions and operations must allow a user to explore and search the expanding information space without becoming mired in details of the implementation. When it becomes necessary to expose some aspect of the implementation to the user, it must be done in a simple way that is semantically integrated with the user's view of the system. Most of the concepts used in the content routing system design are familiar to users of computer systems, as the example of Section 2.1 shows. However, some new ideas are necessary for negotiating a very large information space.

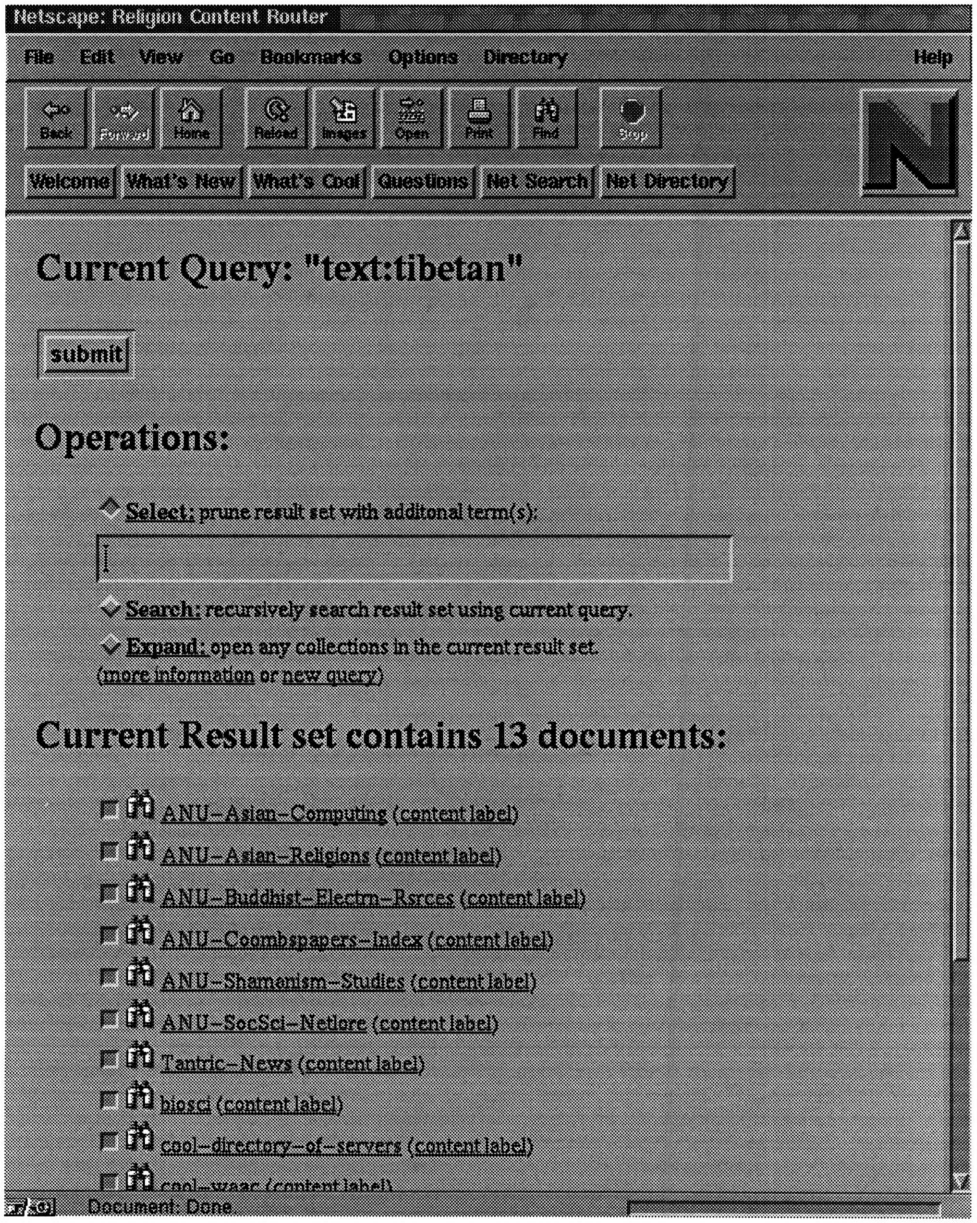


Figure 2-4 Submitting a Query

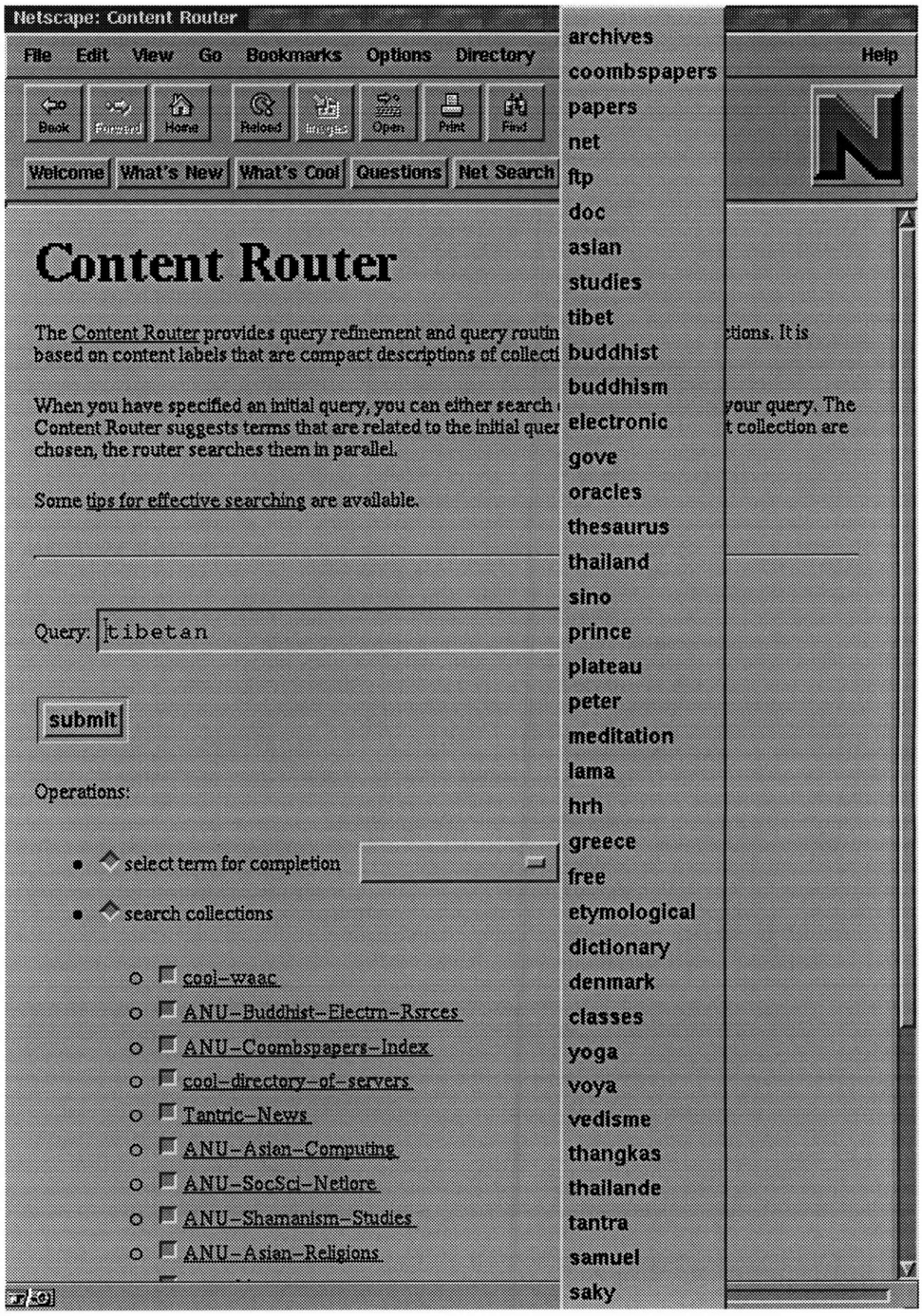


Figure 2-5 Query refinement

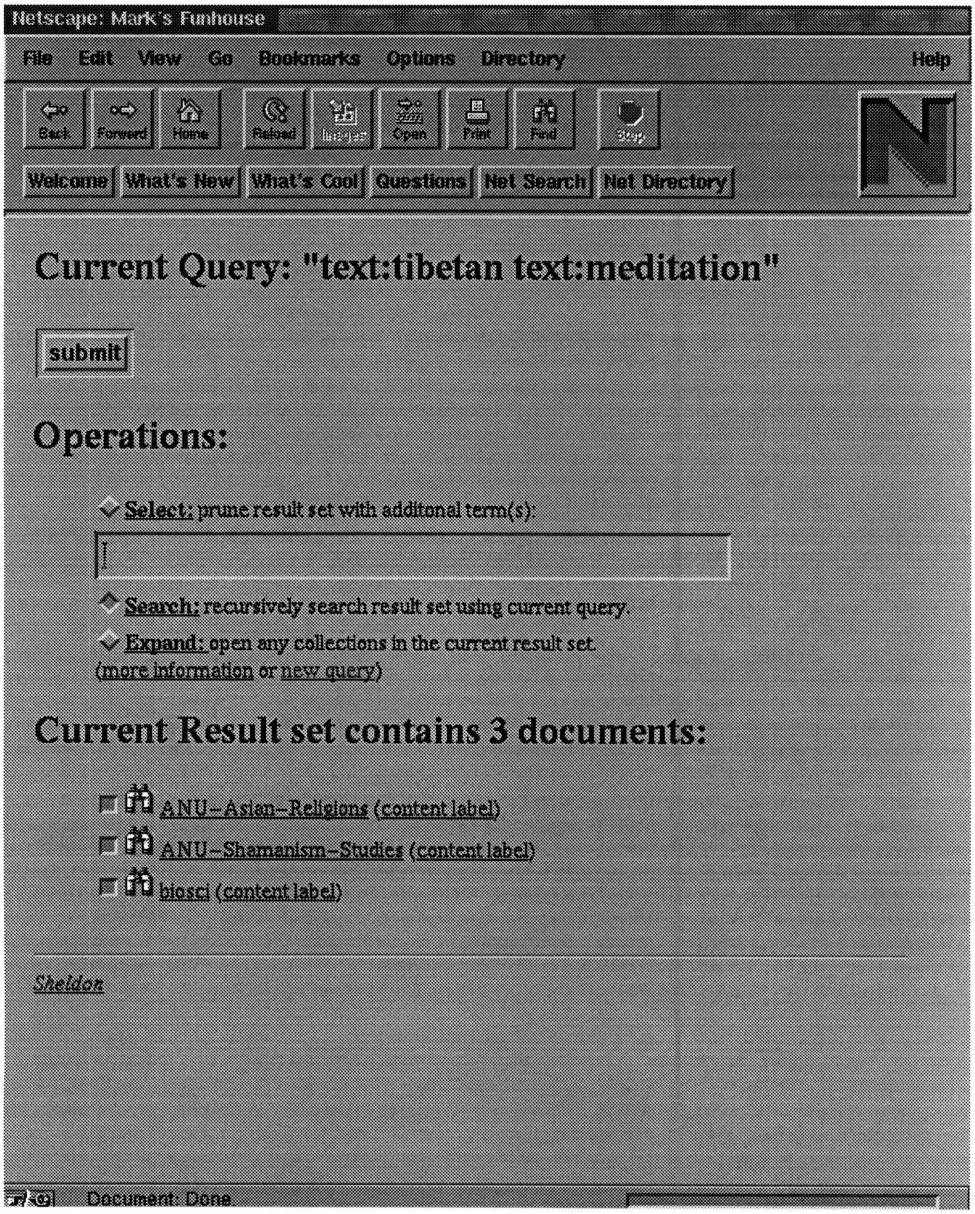


Figure 2-6 Restricted search

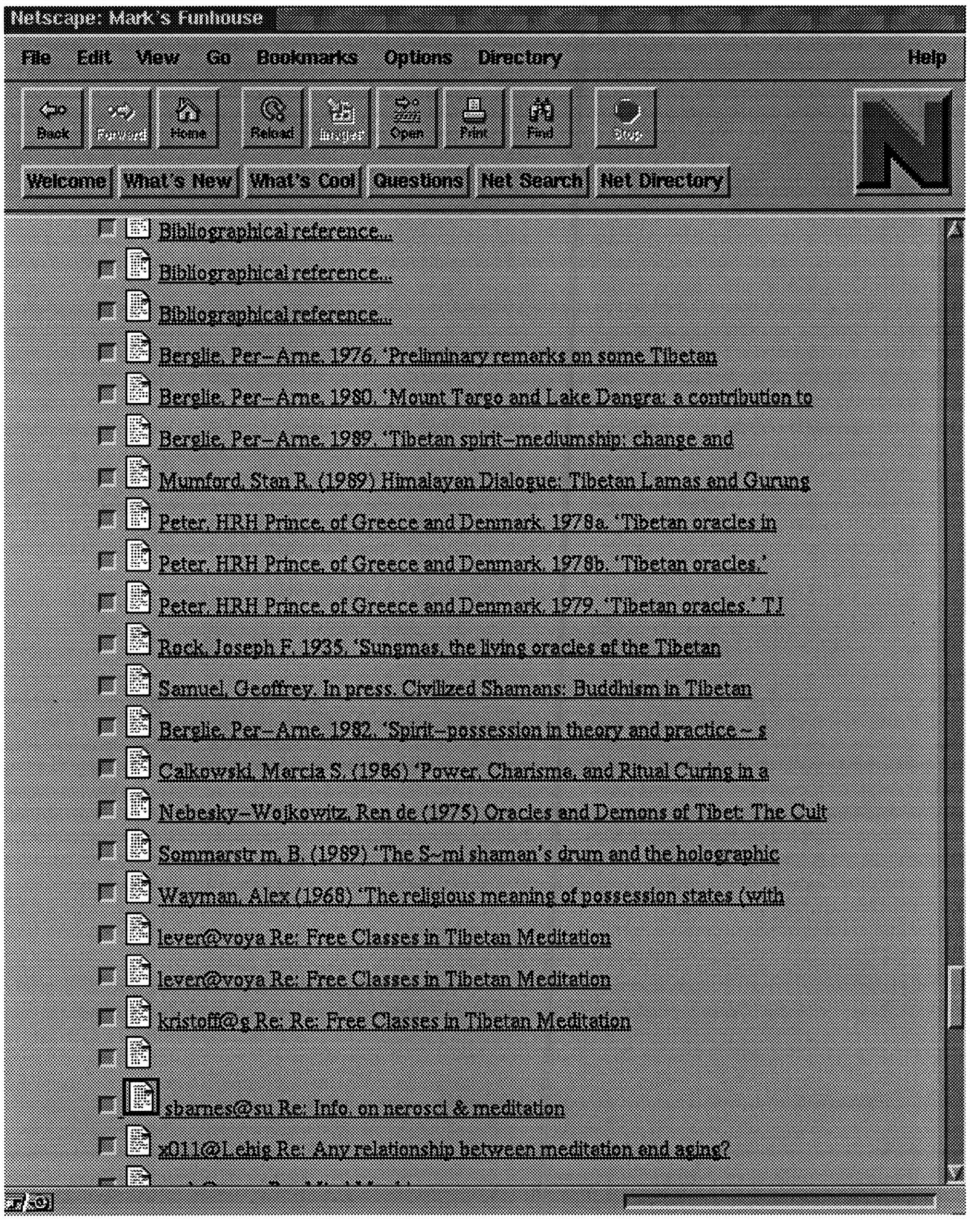


Figure 2-7 Documents

This section describes the abstractions that content routing systems use to organize the information space for the user. Section 2.3 will explore in more detail how these simple concepts can be implemented architecturally, and Chapter 3 will discuss the particular implementations built as part of this thesis work.

Familiar abstractions include *base documents* and *directories*. Base documents are the individual data objects ultimately of interest to the user, including conventional text documents, reports, electronic mail messages, PostScript documents as well as sound and video footage. The set of base document types is extensible and may vary among collections. The content routing system design assumes, however, that queries and, therefore, indexed terms, are textual. Directories are sets of documents, including other directories, which behave like directories in a file system, gopher, or the World-Wide Web.

The key new abstraction of the content routing system design is the *collection document*, or *collection*. A collection consists of a set of documents (possibly including other collection documents) together with a description of the set called a *content label*. Collections help to organize the search space by grouping related documents together into aggregates. Because a collection contains other documents including collections, it can be viewed as a hierarchical arrangement of documents in a directed graph. Figure 2-8 shows a nested structure of collections each with its content label attached.

A content label is a compact representation of the set of member documents in a collection. It is intended that different applications will use different sorts of content labels. Section 2.3.3 discusses the form of content labels in more detail.

As the example of Section 2.1 illustrates, the content routing system uses the notion of a *result set*. A result set is a set of documents that represents the roots of future browsing and searching operations (as Section 2.2.3 will describe). A result set is distinguished from a directory by the fact that it cannot be named, at least with the current interface. Also, it is worth emphasizing the inherent dynamic nature of result sets, which represent the state of a user's session with the system. (However, content routers are free to have

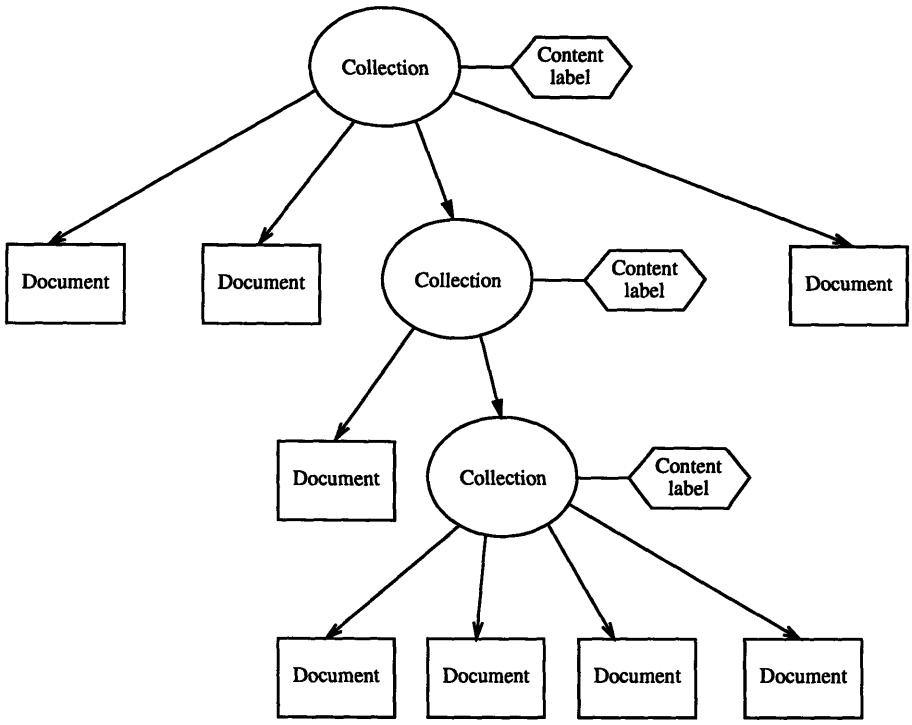


Figure 2-8 A content routing system provides access to a hierarchy of collections.

directories that are dynamically created. This was done in the semantic file system, and initial content routing system implementations used dynamically computed virtual directories for result sets.)

More precisely, a content routing system has the following semantic domains:

$$
\begin{aligned}
\textsc{Document} &= (\textsc{Base-Document} + \textsc{Directory} + \textsc{Collection}) \times \textsc{Name} \\
\textsc{Name} &= \textsc{String} \\
\textsc{Base-Document} &= \text{text} + \LaTeX + \text{video} + \ldots \\
\textsc{Directory} &= \textsc{Document}^* \\
\textsc{Collection} &= \textsc{Document}^* \times \textsc{Content-label} \\
\textsc{Result-Set} &= \textsc{Document}^*
\end{aligned}
$$

where the base documents are an extensible set of document types like ASCII text, video, *etc.* whose representation will be left unspecified; and content labels are as described below. (+ is discriminated, or tagged, union; * is Kleene star; $\times$ is cartesian product.) Notice that result sets are not themselves documents and that they do not have names. Designs with nameable result sets will be explored in future work.

Though directories and collections appear similar semantically, they are distinguished by some operations, described below. The motivation is that directories are intended to be light-weight, local constructs. Collections are typically implemented by remote servers and many interactions will involve communications overhead. This is exposed to the user so that the user can choose operations to balance information services against performance.

2.2.2 Queries

To conduct an automatic search, the user must specify her information needs to the system in the form of a query. The system uses the query to identify components of the search space that are relevant to the user's needs. This section describes the query model used by the content routing system, which is the same simple predicate data model of [22, 21].

For associative access, the content routing system assumes that documents are represented by *attributes*. An attribute consists of a *field* name and a *value*. Two examples of attributes are `text:database` and `author:ullman`.

Typically a server will automatically extract these attributes from documents. For example, a plain text document may be represented by an attribute for each unique word in the document as well as attributes indicating the location and owner of the document; video footage may have associated text annotations or close captioned text.

Directories are represented with attributes based on their names, position in the hierarchy, and owner. Collections are represented by their content labels which contain attributes, though Section 2.3.3 will have more to say about how they are treated. The

semantic file system showed how to build a system based on an extensible set of *transducers* for automatically extracting attributes in a file type specific way [21].

The content routing system leaves the precise semantics of fields to the administrators of information hierarchies and treats field names as strings. We have experimented with some information hierarchies in which certain field names have a fixed, predefined semantics, *e.g.*, the `author` field represented the author of the document. It is also possible for fields to make use of a controlled vocabulary, for example, the Library of Congress catalog system has a fixed set of names for different areas of knowledge. Other fields may have semantics that vary from domain to domain. For example, `category` is a controlled vocabulary field defined by the New York Times wire service [57, Sections 3.1 and 3.2], but may be used differently by a particular library cataloging system. In our experiments with user file systems, individual documents may export their own field names, thus the set of fields is extensible.

Values can assume a variety of types, including integers and dates. Thus far however, the prototype content routing systems have implemented all values as strings.

From the point of view of associative access, then, base documents and directories are represented by sets of attributes (that are extracted from them in an implementation-dependent way), and collections are represented by their content labels. Field names are strings, and values can come from an extensible set of data types:

$$
\begin{aligned}
\textsc{Base-Document} &\propto \textsc{Attribute}^* \\
\textsc{Directory} &\propto \textsc{Attribute}^* \\
\textsc{Collection} &\propto \textsc{Content-Label} \\
\textsc{Attribute} &= \textsc{Field} \times \textsc{Value} \\
\textsc{Field} &= \textsc{String} \\
\textsc{Value} &= \textsc{String} + \ldots
\end{aligned}
$$

where $\propto$ should be read "is represented by," and the treatment of content labels is deferred for the moment.

A *query* is a boolean combination of *terms*. The content router prototypes have used attributes as terms. An example query is:

```
text:database and (author:date or author:ullman)
```

A formal syntax of queries is:

$$\begin{aligned} \textsc{Query} &= \textsc{Term} \mid \textsc{Term}\ \ \textsc{Op}\ \ \textsc{Query} \mid \texttt{not}\ \textsc{Query} \\ \textsc{Op} &= \texttt{or} \mid \texttt{and} \\ \textsc{Term} &= \textsc{Attribute} \mid (\textsc{Query}) \end{aligned}$$

where `not`, `or`, and `and` are the usual logical operations.

It is simple to determine whether a document *satisfies* or *matches* a query, or equivalently whether a query is true of a document. This definition proceeds inductively on the structure of queries. A single term query matches a document if the term appears in the attribute set for that document. If a document is considered to be its attribute set, then the document matches a term if the term appears in the document. A single term query is false with respect to a document if the term does not appear in the document. Then the truth or falsity of a query with respect to a document is the boolean combination of the single term truth values. Section 2.3.3 will discuss how to determine when a collection matches a query.

In many systems today, including WAIS, terms are plain text keywords and queries are sets of terms:

$$\begin{aligned} \textsc{Term} &= \textsc{Keyword} \\ \textsc{Keyword} &= \textsc{String} \\ \textsc{Query} &= \textsc{Term}^* \end{aligned}$$

Keyword-based queries can be mapped into an attribute-based scheme very easily. The content router prototypes use `text` attributes for plain text terms that occur in a document. Thus, a keyword k corresponds to the attribute `text:`k. (This is the strategy the content router prototypes have used when interoperating with keyword-based information servers such as WAIS.)

Keyword-based systems normally use a vector-space model in which a document matches a query if it contains *any* term in the query. Documents are ranked based on how many query terms they contain and on computed term weights. It would be quite reasonable to build a content routing system on a vector model, though, as was mentioned in Section 1.3, it is not clear how to do query refinement in such a system. For more on vector-space models, see [51].

2.2.3 Operations

The content routing system defines a set of eight operations based on the above abstractions that allow a user to browse and search an information space, retrieve documents or content labels, and discover information useful in the formulation of queries. Most of these operations were illustrated by the example of Section 2.1, however, this section contains a complete list of the operations and detailed descriptions of what they do.

A summary of the basic operations of the content routing system appears in Table 2.1. There are three categories of operations: those for browsing, those for query-based associative access, and those for helping in query formulation. The query processing operations *select*, *expand*, and *search* have two entries each because each can be applied to a single document or a set of documents in a result set. Notice that result sets are data objects that can be passed to and returned from operations. Result sets therefore provide a handle for the manipulation of sets of documents and encode the state of a user's interaction with the system. (Some users may find it helpful to think of result

set objects as remote pointers, others may consider them to be like NFS file handles for directories.)

Browsing Operations

The two operations that support browsing allow one to open a collection or retrieve the contents of a document.

Open The *open* operation is used to connect to an initial collection or to browse the contents of a collection encountered during use of the system. In the interface shown in Section 2.1, a user may *open* a collection by clicking on its name. The *open* operation returns the initial, or top-level, result set for the opened collection.

Retrieve The *retrieve* operation is the most familiar of the operations. If the given document is a base document, retrieving it returns the document's contents. Retrieving a collection returns a human-readable version of the collection's content label. Retrieving a directory returns a result set consisting of the documents in the directory.

Query Processing Operations

The three query processing operations provide associative access to the contents of a content routing system. They differ only in their treatment of collections, *i.e.*, they differ in how far down an information hierarchy a search is carried out. All these operations collapse the directory structure of collections, treating a collection's contents as a flat set of documents, though they will only return documents available via the given result set.

Some implementations may estimate the cost of queries and refuse to process those it deems too expensive. The user may be required to narrow the query, manually remove items from the result set before the search, or perform a more limited query processing operation.

All the query processing operations may be applied to an individual document or to a set of documents. The result of applying any of these operation to a set of documents is the union of the results of applying the operation to each member of the set.

open	**open** *collection-name* initializes a connection to *collection-name* and returns the initial result set for that collection.
retrieve	**retrieve** *document* returns the contents of *document*. In the case of a collection, returns a human readable description of the collection's contents (content label). In the case of a directory, returns the contents of the directory.
select	**select** *document query* if *document* is a base document, returns *document* if it matches the query. If *document* is a collection, returns *document* if its content label matches the query. If *document* is a directory, applies **select** recursively to each member of the directory and adds *document* to the combined results if it matches the query.
select	**select** *result-set query* returns the union of the result sets obtained by applying **select** to every document in *result set*.
expand	**expand** *document query* if *document* is a collection, then returns the result of a select at the top-level of that collection. If *document* is a directory, then returns the union of the result sets obtained by applying **expand** to every document in *document*. Otherwise, returns a result set containing just *document*.
expand	**expand** *result-set query* returns the union of the result sets obtained by applying **expand** to every document in *result-set*.
search	**search** *document query* if *document* is a base document, returns *document* if it matches the query. If *document* is a collection and its content label matches the query, then returns the result of a **search** of that collection. If *document* is a directory, applies **search** recursively to each member of the directory and adds *document* to the combined results if it matches the query.
search	**search** *result-set query* returns the union of the result sets obtained by applying **search** to every document in *result-set*.
show-fields	**show-fields** *result-set* returns a list of available attribute field names.
show-values	**show-values** *result-set field-name* returns a list of potential values for *field-name* attributes.
refine	**refine** *result-set query additional-args* returns a list of recommended query terms that may be used to reduce size of the search space.

Table 2.1 Router operations for browsing, query processing, and query formulation

Select The *select* operation is the most restricted search provided, and, therefore, the coarsest in granularity. A *select* returns the documents in the universe implied by the given the result set that match the query. `Select` considers the contents of directories recursively, however, it does not look inside collections that may appear in the universe. Thus, a select at a node in Figure 2-8 returns only the children of that node that match the query.

Operationally, a *select* applied to a particular *document* and *query* proceeds thus: If *document* is a base document, then *select* returns a result set containing *document* if *document* matches the query, an empty result set otherwise. If *document* is a collection, then likewise returns *document* if its content label matches the query. If *document* is a directory, then return *document* if it matches the query *plus* the union of the results of applying *select* to each document in the directory. In other words, *select* returns matching documents while flattening directories.

Expand The *expand* operation is like the select operation except that it expands one layer of the collection hierarchy, providing somewhat more detail but without the expense of an exhaustive search. Thus, an *expand* at a node in Figure 2-8 results in all the matching non-collection children of that node plus all the matching grandchildren of the node (available via matching collections). An alternative operational description is this: Perform a select operation with the same arguments. All non-collection documents in that result will be in the *expand*'s result. For each collection in the intermediate result, perform a *select* at the top-level of the collection. Combine all the results.

Search The *search* operation treats the entire search space reachable from the present result set as if it were a single, flat, information system. Thus, referring again to Figure 2-8, a *search* operation will return all the non-collection documents that match the query in the graph rooted at the node from which the search commenced, proceeding through collections that match the query. Operationally, again, perform a *select* at the present collection with the same result set and query. Each non-collection document will be returned as part of the result. For each collection returned from the `select`, perform

a *search* at the top-level of that collection. Combine all the results.

Query Formulation Operations

The query formulation operations help the user learn how to formulate queries and help manage the complexity of the search space. A new user may know nothing about how a collection indexes its documents. Thus, the user needs a way to find out what terms are available for queries. Retrieving sample documents is one way to do this (and the only way on previous systems), but it is valuable if the system provides special help. Once the user gets started, a typical session proceeds with the user alternately broadening and narrowing queries. Typically users start with broad queries. The set of documents to peruse becomes smaller as a query is refined. The user refines queries either by using system recommended completions or by using attributes discovered in interesting content labels or by other system-supplied resources. When a query is sufficiently narrowed, its collection documents may be expanded. This process continues in a pattern of contracting selections alternating with expansions with a final search in a reduced space of documents. In addition to these operations, the content routing system allows users to browse the system and look inside documents and content labels to learn about available terms.

Show-fields The *show fields* operation displays the set of attribute field names known to the currently open collection(s). Ideally, the set of fields displayed should be limited to those found in the documents of the current result set. Note that *show-fields* does not require an outstanding query, and can thus be used when connecting to a new collection to learn about its contents and how they are indexed.

Show-values The *show values* operation lists possible values for a given field. Ideally, the values displayed should be selected from those found in the documents reachable from the current result set. Implementations may want to recommend values that reduce the search space (as in *refine* below). Note that *show-values* does not require an outstanding query, and thus be used when connecting to a new collection to learn about its contents and how they are indexed.

Refine The *refine* operation takes a given query and returns suggested query terms (attributes) that will reduce the search space if they are added to the query. Section 2.3.2 will provide more detail on how this may be done.

2.3 Content Routing System Architecture

The content routing system design presented above must be realized by a concrete systems architecture. The goal of the architecture presented here is to realize the functionality above as simply and transparently as possible. This section is a high-level description of the content routing system architecture. It will not be concerned with the details of underlying network protocols, *etc.* Chapter 3 will describe prototype systems that use the Semantic File System/Network File System (which is RPC-based) and the Hypertext Transport Protocol (which is TCP-based).

Architecturally, then, a content routing system is a network of information servers. Leaf nodes in the network are end-point information servers that store conventional documents. The hierarchical network of internal nodes is composed of *content routers*, which are information servers that support collections and the set of operations listed above. Figure 2-9 shows a content routing hierarchy. In such a system, *expand* and *search* operations cause queries to be routed to relevant information servers for processing (the downward-pointing arrows in the figure). If an operation is forwarded to multiple servers then the results are merged. A content router may elect to forward operations to servers or to refuse to do so depending on the cost effectiveness or expense of that operation. Content label information flows upward from the leaf servers as each server registers with its parent.

The structure of a content routing network is isomorphic to the collection hierarchy seen by the user of that hierarchy. For example, Figure 2-10 implements Figure 2-8. Each collection is implemented by an information service for the collection's component documents plus a content label that describes the collection. Each content router (an

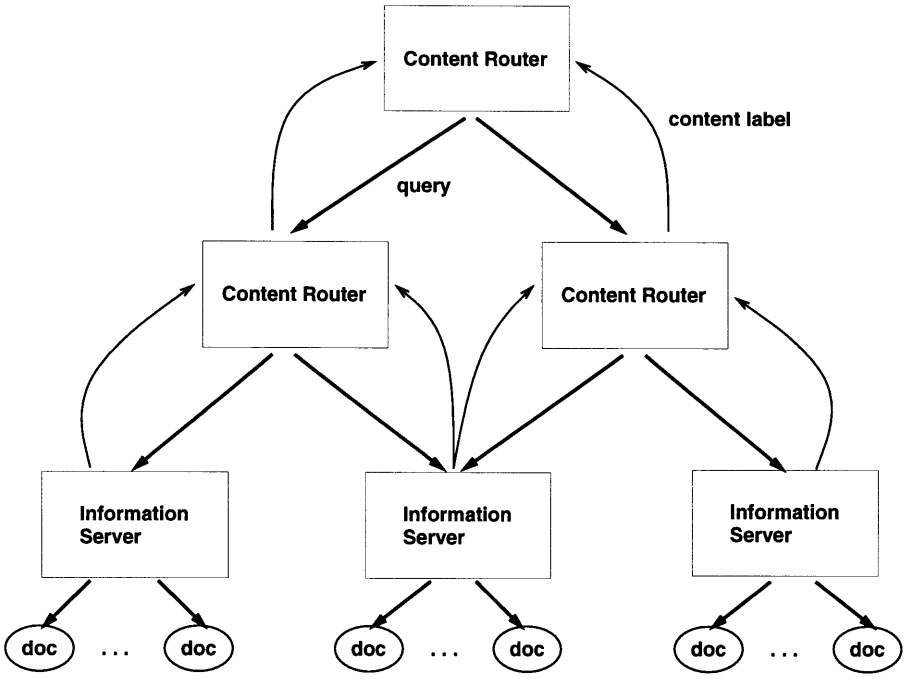


Figure 2-9 A content routing system is a hierarchy of information servers.

information service that provides access to collections) stores any base documents and directories it exports plus the content labels for any collections registered with it. Servers are programs that may or may not reside on the same physical host computer. Collection containment (shown by the graph connectivity in Figure 2-8) corresponds to server registration (shown by the server connectivity in Figure 2-10.

Some of the operations described in Section 2.2.3 are very simple to implement. For example, in an HTTP-based implementation, registration consists in a subsidiary server sending URLs for its top-level and for its content label. An *open* operation would consist in retrieving the top-level document, and other operations would consist in sending and receiving form documents of the appropriate form. See Chapter 3 for more details.

However, there are two key services that are needed to implement the semantics given above: query routing and query refinement. *Query routing* is the process of identifying relevant servers for a user's query, forwarding the query to those servers, and merging the results. *Query refinement* has been defined above. These two services are described below

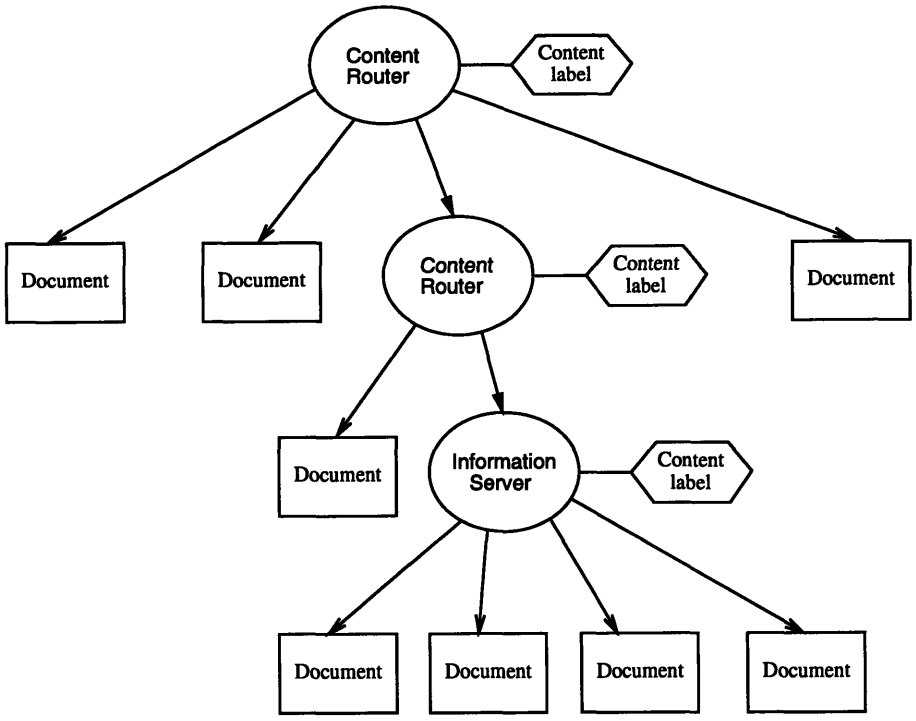


Figure 2-10 The server topology is isomorphic to the collection topology.

in Sections 2.3.1 and 2.3.2, respectively. Their precise implementations are described in Chapter 3.

In addition, the content routing system architecture is parameterized by the choices of the contents and semantics of content labels as well as the structure of information hierarchies. Section 2.3.3 discusses the choices for content labels, and Section 2.3.4 discusses how to build information hierarchies. The detailed choices used in the experimental work of this thesis are given in Chapter 3.

2.3.1 Query Routing

One of the goals of this research is to establish whether it is possible to route queries to a meaningful subset of a large collection of information servers. This section gives the approach to this problem taken by the content routing system. Experience with an actual implementation appears in Chapter 3.

To conduct a search for items matching a given query, a content router identifies relevant documents in its collection, identifies which of those documents are collections, forwards the query to the corresponding servers, and merges the results for presentation to the user. Some formalization will help to describe exactly how a content router performs this task.

Recall that a user's query Q is a boolean combination of terms. Terms may be either attributes (field name, value pairs) or keywords. Q can be considered as a predicate that is either true or false of a document, *i.e.*, $Q(d)$ is true if and only if document d matches Q. A collection document matches a query if its content label matches the query. Section 2.3.3 will give two alternative ways to define when a content label matches a query. Section 2.2.2 defined what it means for a base document of directory to match a query.

The *document space* for a query Q represents all the documents in end-point information servers that matches the query. Ultimately, this is the set that interests the user. It is defined by:

$$\mathcal{D}(Q) \equiv \{d \in U |\ Q(d)\}$$

where U contains every (conventional) document reachable by traversing the network from the current content router.

Identifying relevant documents in the collection is a standard information retrieval operation. Determining which collections are relevant to a query is described in Section 2.3.3. This information is exactly what is necessary for a *select* operation. This will also automatically identify the set of servers to which a query should be routed, the *route set* for the query.

More formally, the *route set* for a query Q is

$$\mathcal{R}(Q) \equiv \{d \in C |\ Q(d)\}$$

where C is the set of collection documents available on the current server.

The expand operation then simply returns the set of documents at these remote servers that match the query:

$$\text{EXPAND}(Q) \equiv \{d' \in d |\ (d \in \mathcal{R}(Q))\ \wedge\ (d' \in d)\}$$

The *search* operation simply applies the rule for the expand operation recursively and returns only base documents. That is, *search* is equivalent to the transitive closure of *expand*.

$$\text{SEARCH}(Q) \equiv \text{EXPAND}^*(Q)$$

The content routing system uses this as its approximation of the document space of Q defined above. How close this approximation is depends on how good the content labels are. For example, if content labels contained all indexed terms, then the content router would have an effectively complete index, and the approximation would be perfect. Of course, this will not scale, and the system must trade off the quality of the approximation for scalability.

Note that this description assumes that the hierarchy is cycle free. If there are cycles, then there are simple ways to break them so that query routing does not loop indefinitely. A simple strategy is to encode all the collections a query has been routed through in the arguments to the search operations. If a collection finds itself in the list, it simply terminates without performing the operation.

2.3.2 Query Refinement

An important goal of this research is to establish whether a system can automatically suggest improvements to user queries in order to help the user cope with the vast information space. This is important because, in a very large distributed set of information providers, naive queries can be prohibitively expensive to process and will overwhelm the user with irrelevant material. This section explains the principles behind the approach taken by the content routing system. Query refinement is a unique feature of this work.

This project has produced the first large-scale system that automatically guides the user in the formulation of better queries. Experience using the techniques of this section in working systems suggests that query refinement is an invaluable tool in a large distributed information system and that it can be efficiently implemented (see Section 3.3 for details of a particular implementation).

The approach taken to query refinement here is different from the prior efforts in the Community Information System [22]. In the Community Information System, databases were described by content labels consisting of small queries which were true of every document in the database. A simple theorem prover required the user to choose query terms that guaranteed the query could be completely satisfied at available news wire databases. The approach described here is statistical in nature, using conditional probabilities of term collocation (see below). I feel that the approach described here is better suited to the larger-scale, more heterogeneous environments of modern global networks where it may be difficult or impossible to organize information servers so they are authoritative for precisely specified generator queries.

The task of query refinement is to generate a list of terms related to a query that can be used for formulating new queries that reduce the search problem. The search problem may be reduced either by reducing the set of documents that match the query (the *document space*) or by reducing the number of servers that are relevant to the query (the *route set*). The route set approximates the set of servers that contain any documents that match the query (see Section 2.3.1).

Implementations may use a thesaurus for term recommendations, though the implementations described in this thesis have not done this because there is already substantial work in this area. The traditional use of a thesaurus is for *query expansion*, *i.e.*, to increase recall. The technique is to replace a term with a disjunction of terms from its thesaurus entry (or adding terms to the query in a vector model) [51]. This is antagonistic to the goal of reducing the result set size. However, it is valuable to use a thesaurus in conjunction with other means of producing suggestions for query modifications, es-

pecially in a system where the user may select the degree of association between the query and the suggested terms (see below). One can use the techniques of conditional probabilities of term collocation together with clustering techniques to build a kind of thesaurus automatically. For that matter, using different ranking functions, one could construct off line associations of words ranging from synonym lists (like a thesaurus) to antonym lists. For related work on *query expansion* (increasing the result set size) and use of thesauri, see [51, pp. 75–84] as well as [64, 48, 15].

Conditional Probabilities of Term Collocation

To determine what terms are related to a query, the query refinement algorithm uses the conditional probability that one term is collocated with another. In other words, the system poses the question: given the documents that match a particular term, what is the probability that some other particular term occurs in one of those documents? Terms whose occurrences are highly correlated are assumed to be related.

The system, therefore, uses the conditional probability of term collocation to recommend terms that are related to a given query and that efficiently partition the document space. The conditional probability p_i that term t_i occurs in a document that satisfies Q is the size of the document space for Q conjoined with t_i divided by the size of the document space for Q:

$$p_i \equiv \frac{\|\mathcal{D}(Q \wedge t_i)\|}{\|\mathcal{D}(Q)\|}$$

If a term has a high conditional probability p_i, then it is statistically related to the query, and therefore likely to be semantically related as well. A term with a low conditional probability will more dramatically reduce the size of the document space when it is conjoined with Q.

The system's term recommendations for the *refine* operation are an ordered set, or tuple, of terms. The list is ordered by some ranking function, $\mathcal{E}$ (see Section 2.3.2 below).

$$\textsc{Refine}(Q) \equiv \langle t_0, t_1, \ldots, t_n \rangle$$

where $\mathcal{E}^Q(t_i) > \mathcal{E}^Q(t_{i+1})$.

A content label must contain, therefore, not only the terms exported by a server, but also information about term collocation. We are experimenting with various ways of representing this information in content labels. However, since content labels do not contain all terms available in a database, the content router does not have perfect knowledge about term collocation.

The prototypes described in Chapter 3 simply offers the user the 40 terms with the highest (approximated) conditional probabilities. See Section 3.3 for details of the algorithm for computing these terms as well as a description of how well it performs.

Note that this technique may be applied to reducing the route set, $\mathcal{R}$, as well as the document space by computing the conditional probability thus:

$$p_i \equiv \frac{\|\mathcal{R}(Q \wedge t_i)\|}{\|\mathcal{R}(Q)\|}$$

So far, our experience is that query refinement based on the document space produces semantically meaningful suggestions and also effectively reduces the size of the route set. Query refinement based on reducing the route set, using term collocation in whole collections, has not proven helpful. In our prototypes, documents are not sufficiently tightly clustered on servers. Finer grained information is necessary. We are investigating a compromise strategy that uses clustering within collections.

Ranking Functions

In general, a user may want to learn about terms closely correlated with the query (synonyms), terms inversely correlated with the query (antonyms), or terms moderately correlated with the query. Synonyms and antonyms provide feedback on what sorts of terms are indexed and help the user formulate better queries. Moderately correlated terms are especially useful in reducing the document space by a reasonable amount while not eliminating documents so quickly that highly relevant items are lost. The system uses a *ranking function* to order terms and suggests to the user a manageable set consisting of

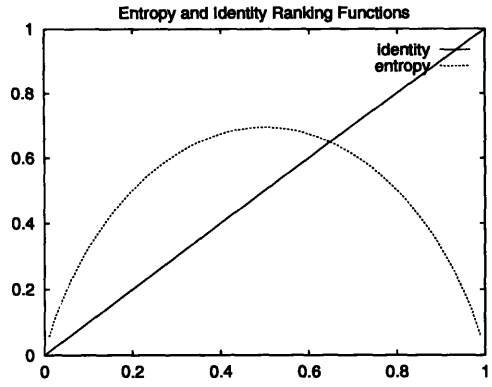


Figure 2-11 Entropy and Identity Functions

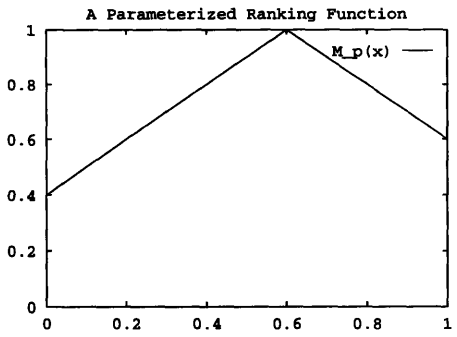


Figure 2-12 $\mathcal{M}_{0.6}$ Function

the highest ranked terms. For a more detailed discussion of ranking functions in general information retrieval applications, see [51, pp. 59–71] or [27, pp. 363–376].

Entropy functions can be used to give favorable ranks to terms that are moderately correlated with the query terms; that is entropy-based ranking functions prefer terms that reduce the result set by a moderate amount. These function represent a measure of the information content of the terms. A generalized entropy ranking function has the form

$$\mathcal{E}(t_i) = p_i F(p_i)$$

There are a variety of choices for F. One standard information theoretic entropy function, $\mathcal{E}_S$, is

$$\mathcal{E}_S(t_i) = -p_i \log p_i$$

Figure 2-11 illustrates the behavior of $\mathcal{E}_S$ and the identity function, which assigns each terms conditional probability as its rank. As the figure shows, $\mathcal{E}_S$ favors terms with probabilities closer to 0.5.

It is useful to have a ranking function that is parameterized by the most favored probability (or result set size reduction). This would allow the user or the system to rank terms differently depending on the context. For example, the ranking could suggest synonyms or antonyms. A simple ranking function $\mathcal{M}_{\overline{p}}$ that is parameterized by the

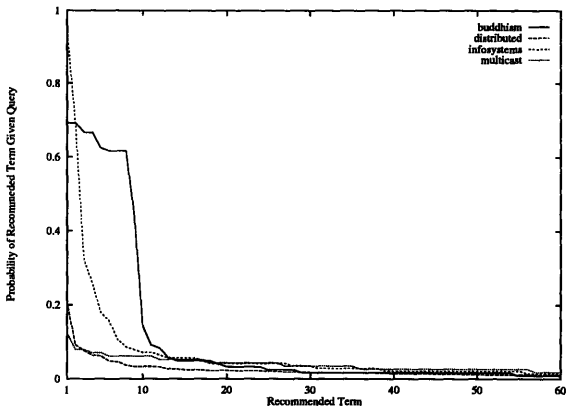


Figure 2-13 Refinement Probabilities

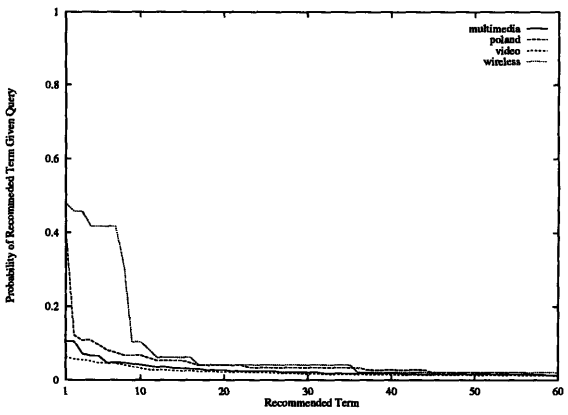


Figure 2-14 More Probabilities

favored conditional probability, $\overline{p}$ is

$$\mathcal{M}_{\overline{p}}(t_i) = 1 - |p_i - \overline{p}|$$

Figure 2-12 illustrates the behavior of $\mathcal{M}_{\overline{p}}$ for $\overline{p} = 0.6$.

The system also should consider the expected time it will take to evaluate a query. This in turn depends on the number of remote servers involved in servicing the query, *i.e.*, the size of the route set. We are currently investigating techniques for producing cost estimates.

We have found the use of the identity ranking function to be acceptable in the prototypes because, in the experimental data, the terms most frequently collocated with a query typically have very small probabilities. Figures 2-13 and 2-14 show some anecdotal evidence of this. The figures show the conditional probability of each of the top 60 candidate terms for suggestion. The highest conditional probabilities are often under 0.5, and the conditional probability is always under 0.3 by the tenth best term. Thus, there are few high probability terms to eliminate with a ranking function. Since we take the 40 highest probability terms, we eliminate the very low probability terms. A good ranking function may change the ordering of the top 40 terms, but is unlikely to have a significant impact on the terms in the set. This situation may change as the implementation gains

more detailed knowledge of query term collocation.

2.3.3 Constructing Content Labels

This research seeks to define metadata structures that can be used to organize the information space and provide other assistance to users. It is also important to be able to extract these metadata descriptions of server contents from large numbers of disparate, incompatible servers. These metadata structures, content labels, are used to describe collection contents to users, to route queries to relevant servers, and to provide query refinement suggestions. Content labels, are therefore at the very heart of the content routing system, and their structure and semantics profoundly affect the utility of the system. In addition, content labels provide a focal point for system administration: size restrictions affects scaling, content restrictions determine the look and feel of the system.

Content labels represent a compromise for a scalable architecture. The ability to control content labels is what enables a content router to avoid unlimited demands on its resources. Limiting content label size not only helps manage local resources, but also encourages administrators of remote information servers to make effective use of their content label advertising budget. A budget scheme will give priority to specialized servers when users submit more specific queries.

The remainder of this section describes two approaches to content label semantics, discusses considerations for including terms in content labels, explains an approach to extracting content label data from a particular type of incompatible information server, and surveys issues in supporting query refinement. See Section 3.1 for a detailed look at the use of two techniques for the automatically constructing content labels.

Content label semantics

The content routing system design does not limit the form of content labels. There are at least two strategies for building content labels: generator queries and surrogate documents.

Generator Queries In this approach, which was used in the Community Information System [22], a content label is a query that is true of all documents in the collection. In principle, such a content label could consist of the disjunction of all of the terms that appear in a collection. Alternatively, a content label could simply be the name of the host that contains a set of files. In practice, content labels will lie between these two extremes, containing terms including host names, domains, authors, libraries, and subjects.

$$\text{CONTENT-LABEL} = \text{QUERY}$$

If content labels are generator queries, then a collection c matches a query Q if $Q \rightarrow \text{CONTENTLABEL}(c)$. Logical implication is an elegant mechanism, and simple theorem provers can be used to identify relevant collections and to extract terms that can be used for query refinement (see [22]).

Generator queries are a good choice for small-scale or highly constrained systems, such as name services, where a collection's contents is completely described by a query. That is, the generator query is true of all documents in the collection, and all documents implied by the query are in that collection. This represents a kind of closed world assumption [50]. Unfortunately, in a large network of more general information servers, it is very hard to guarantee this closed world assumption.

For example the following very small generator query content label implies that every document of the collection has attribute `subject:database` and `collection-type:software` and is either in the `cmu.edu` or the `stanford.edu` domains.

```
[ (subject:database) and ((domain:cmu.edu) or (domain:stanford.edu)) and
(collection-type:software) ]
```

Surrogate documents In this approach, a content label is a surrogate document for a collection containing terms culled from that collection's contents. Such a content label may be viewed as an advertisement for the collection. The administrator of the

collection may use various tools to select terms for the content label, computing a centroid document [51] for example. A surrogate document content label is just a set of terms (which is what an abstract document is to the query processing part of the system), and thus it may be indexed, like any base document, under the terms appearing in it. On may think of a surrogate document as being like a generator query that consists of a disjunction of the terms in the surrogate. A collection c matches a query Q if it matches the query as an ordinary document.

$$\text{CONTENT-LABEL} = \text{ATTRIBUTE}^*$$

The surrogate document approach is easy to implement. Content labels are indexed like any other documents. The content labels that match a query are computed just as are the other documents that match the query. In fact, the content label information may be stored in the same index structures, as it is in the prototypes built as part of this thesis. It is still necessary to identify which documents are content labels so that the system can determine where to route queries, *etc.* One can use a simple mechanism like storing content labels in files with unique suffixes. Then, the content labels are the members of a query result that have the unique suffix. Content labels may also be indexed with a special attribute that identifies them as content labels.

The surrogate document approach, though it has the advantages of simplicity and uniformity, does not have the strong semantics of the generator query approach. In particular, if a generator query is not implied by a query, then there are no documents under the corresponding collection that match the query. This is a result of the closed world assumption. In a surrogate document, some terms indexed in documents within the collection may not be in the content label, and thus will not be found via the route set approach used for query routing. However, if a user does look within that collection, more terms will become available. (See below for some thoughts on propagation of terms

in content labels.)

A surrogate document content label for a server that exports the work of certain authors might begin as follows:

```
author:gifford author:sheldon author:otoole author:jouvelot author:duda
author:weiss author:reistad text:content text:routing text:system ...
```

Selecting terms for content labels

A good content label will contain two kinds of attributes:

- Administratively determined *synthetic* attributes that describe a collection but may or may not appear in the documents themselves. For example, an administrator may want to advertise that a server has the attribute `collection-type:software`.

- Attributes automatically derived from the collection contents, possibly using various statistical tools. For example, frequently occurring descriptive terms are good candidates for content labels. These attributes are called *natural* attributes because they arise naturally from the data.

A particular hierarchy may impose constraints on the propagation of attributes in order to get back some of the benefits of the closed world assumption. For example, a hierarchy may want to guarantee that, if a field is available for queries, then the server is authoritative for that field. Essentially, one must propagate *all* values of a field or none at all, along with information about which fields have been dropped. Then a server could determine when there is no possibility of any document matching a query beneath a particular node in the hierarchy. Such schemes are left for future work. (I am grateful to Ken Moody and Jean Bacon of Cambridge University for input on this subject.) On the other hand, for regular documents, this is not a practical approach because it is not feasible to propagate all `text` attributes, since these represent the vast majority of attributes, and the system is not useful without any `text` attributes. It is also very hard

to guarantee that a server is authoritative for some attribute if autonomous servers are constantly being added to the hierarchy.

A particular content routing hierarchy may restrict certain attributes, fixing their semantics, limiting the choice of values, or even using a controlled vocabulary. These are same issues discussed with respect to queries above in Section 2.2.2. In addition, a hierarchy may predefine and/or require certain attributes be present in content labels. For example, it may be necessary to include `hostname`, `hostaddress`, and `administrator` fields.

Information providers must choose what terms to include in their content labels bearing in mind that the data will be used by a content router to determine whether their databases are relevant to queries. Content labels must therefore reconcile two conflicting objectives. On one hand, they must contain terms that characterize the collection as a whole (typically high frequency terms), and on the other hand, they must be able to distinguish servers with respect to each other. That is, they must be concerned with advertising and with discriminatory power. It is possible to divide this task between an information provider and a content router with which it registers: The information provider may supply a broad content label with many high frequency (or highly significant) terms, and the content router may decide to keep only terms with adequate discriminatory power. There are standard metrics for computing the discriminatory power of terms based on their frequencies of occurrence [51]. For experimental data on some sample information servers, see Section 3.1.

Terms may be chosen based on their statistical significance or based on some other analysis of their importance, *e.g.*, terms that appear in titles or abstracts may take precedence over others.

Extracting data from incompatible servers

There are two approaches to extracting data for content labels. In the first, an information provider computes its own content label using whatever statistical tools and

information it chooses. This approach has the advantage that it provides an abstraction barrier: the information provider can control exactly what information it releases about itself. It also supports scaling because the work for the collection is done locally at each server. A second approach involves a content router extracting the raw data from the information provider and computing a content label on its behalf. This approach has the advantage that the content router can optimize the information in the content label based on knowledge of the other servers registered with it. The disadvantage is that it must gain access to raw data at the information providers and provide the computation and space necessary to build the content labels. It may be that much useful information is not exported from the information providers, in which case, the quality of the content labels will be severely compromised.

Analyzing term frequencies requires access to the raw index data of an information server. That is why the content routing architecture assumes the first strategy above, that is, it leaves the construction of content labels to the servers with the relevant data. However, to to interoperate with existing servers, our experimental systems have used both approaches. Having the content routers compute content labels has allowed the prototypes to integrate Wide Area Information Servers (WAIS) [29] into the content routing system. This has allowed the research to experiment with far more data, more servers, and with a much wider distribution than would have been possible otherwise. However, there is no direct access to the remote WAIS indices. We were therefore confronted with the need to automatically construct content labels without detailed analyses. Nevertheless, the automatically produced content labels for WAIS servers have proven surprisingly useful.

The information that can be extracted depends on the type of system. WAIS systems make two sorts of content information available to clients: *source* and *catalog* files. These can be programmatically extracted from most WAIS databases by sending a null query. The source file is a brief description of the server, including how to connect to it, and often contains a list of keywords. The catalog file contains a list of all the *document*

headlines (usually titles or subjects) for the database. The headline data has proven very useful because titles are usually chosen to contain words that users will know about and that are very significant in the documents.

A simpler problem to solve concerned WAIS's use of keyword-based indexing. The content routing system simply interposed a gateway between each WAIS server and the rest of the system. Keywords were translated into corresponding `text` attributes. Chapter 3 provides a detailed description of our construction and use of content labels for WAIS.

Supporting query refinement

Content labels must also contain information for query refinement. For the query refinement approach of Section 2.3.2, the system needs an approximation to the document space for a query. This necessitates some representation of term collocation.

The prototypes used a very simple strategy placing collocated terms on the same line of the content label, as if each line represented a single document. Terms could be dropped by filtering them out of the content label, and lines could be dropped. Only terms appearing in document titles were used. This strategy allowed the use of compressed WAIS catalog files, which gave the headline of each document in the collection. Query refinement based on this data has been effective, as the examples in this thesis show. (See Figure 1-5 and 2-5 and Tables 3.3 and 3.4.)

Ongoing research is focusing on document clustering, essentially grouping documents into larger virtual documents for purposes of doing query refinement. If the documents can be clustered so that they are closely related, then terms in the cluster are likely to be related (in the same way that terms in individual documents are related). This produces substantial space savings, but there is not yet enough experience to report on.

2.3.4 Hierarchies Organize the Information Space

Hierarchies are important because they help organize the information space for the user by allowing the manipulation of portions of the information space at varying levels of granularity. In addition, hierarchical data organizations address the scaling issue. No one server must contain global information that will strain its resources and quickly become out of date.

The construction of information hierarchies is not only of technical importance in the design of a content routing system: it is at the very heart of a revolution in publishing and library services. The structure of an information hierarchy represents new methods of publishing (placing a document on a server), dissemination and advertising (registering a server within the hierarchy). See the market-based paradigm for construction of hierarchies described below for one interpretation. Many authors have pondered what digital publishing and digital libraries mean for the interests that participate in paper publishing, advertising, *etc.* See for example [54, 18, 69, 33, 31, 68].

In a content routing system, the roles of editor and publisher are reinvented by the negotiations among authors, information providers, and content router managers (as well as any higher authority that may establish an entire hierarchy under its control). These negotiations will create the rules that keep content consistent, ensure quality, and establish paths to the information consumer. The content routing architecture does not set limits on the rules that may be adopted. Rather, it provides an architecture that is flexible enough to support a broad spectrum of possibilities from tight control to chaos.

Thus, there are two sets of issues in the construction of information hierarchies. First, there are the administrative issues surrounding the choice of the organizing principles of a hierarchy. Second, there are the issues involved in the mechanics of server registration.

Techniques for Information Organization

As stated in Chapter 1, no single hierarchical organization of information can satisfy every information need. Thus, a variety of hierarchical organizations ought to be made

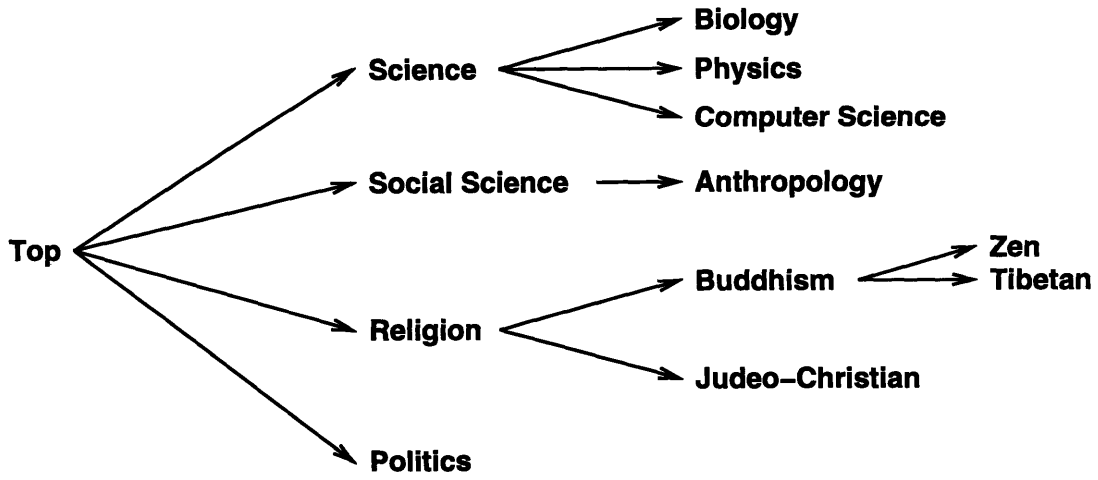


Figure 2-15 A sample subject-based hierarchy.

available. In general, the information available via a network will be organized in multiple, overlapping hierarchies, producing a mesh of trees. Examples of useful paradigms for organizing information hierarchies follow.

A hierarchy may be organized by *administrative domain.* For example, the servers at the MIT Laboratory for computer science may be clustered together into a collection which is itself one of many collections available via a general MIT server. These sorts of hierarchies are essential for businesses concerned with keeping track of their own resources. They can also help users who know the organizational affiliation of the producer of a document.

A hierarchy may also be organized by *geographical location.* This is the approach taken by the X.500 naming system [10]. In addition to name services, geographical hierarchies can be useful for finding copies of a document that are physically nearby or for finding documents whose location is known (*e.g.*, the user knows the author is from Britain).

Subject hierarchies have a long history in library science. The Library of Congress catalog scheme is a good example. These are especially useful for browsing, especially if one knows the subject classification scheme well. An *ad hoc* subject hierarchy is shown in Figure 2-15.

Market-based hierarchies are likely to be very important in the information marketplace of the future. It might be useful to think of a content router as a kind of magazine that targets a particular information market. For example, an entrepreneur might decide to exploit an unfilled need for information on do-it-yourself home renovations. After doing market research to assess the potential for subscribers, the entrepreneur would then gather information and try to get other information servers to advertise in (register with) his content router. There might be a limit on content label size (limited advertising space), or there might be a charge for additional space. Market-based schemes require adequate payment and authentication schemes which are just now beginning to become available. See [62, 38] for descriptions of two Internet-based payment and authentication services being developed at public institutions. A private company is already positioning itself to sell payment and authentication infrastructure in the form of *payment switches* [23].

Such information services are a certainty in the near future. The proliferation of commercial services on the net has already been noted in [40, pp. 31–33]. In 1992, the commercial (`com`) domain was by far the fastest growing Internet domain [56]. In fact, Open Text is already launching a commercial global index of the World Wide Web [41].

Data-driven hierarchies are an interesting alternative to all the above schemes. Documents are grouped into clusters based on an automatic analysis of their contents. These clusters may in turn be clustered, and so on until a there is a manageable number of document clusters. First level clusters are end-point information providers, and clusters of clusters are content routers. There is a vast literature on document clustering, but the Scatter/Gather [11] project has focused on the automatic techniques for clustering very large corpora of documents both statically and dynamically. I am currently investigating the feasibility of experimenting with this strategy.

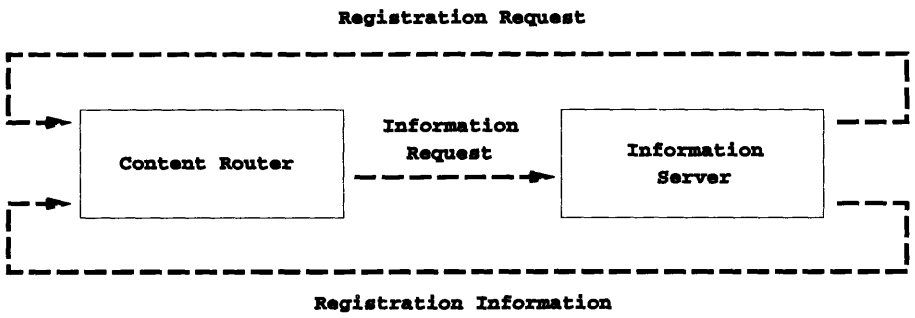


Figure 2-16 The Registration Process

Registration

To allow easy integration of other servers into a network of content routers, the system must support a simple and flexible procedure for information servers to register with content routers. It is very important that the registration procedure allow for human intervention and negotiation of terms, because registration is the point where quality control and resource requirements meet. Content routers may refuse to register servers that do not abide by semantic restrictions or that require too much space or that have overly general content labels. Servers may refuse to register with a content router unless at least a certain portion of its content label is allowed or if it is already unable to support the demand for its resources.

An information server registers with a content router by providing a content label and an interface for the content router operations. The precise mechanisms and syntax involved depends on the implementation technology. For example, a Semantic File System-based implementation would use file system operations, special virtual directories, and queries to conduct the protocol. A World-Wide Web-based implementation could use special forms. However, there are some general features that ought to be present in any registration protocol. This section outlines a simple, abstract registration protocol.

Any registration protocol ought to allow either the content router or the registering information server (which may itself be a content router) to initiate the process. This way,

a content router that becomes aware of a relevant information server through automated or human efforts may ask that server to register; and an information server that would like its information to be reachable from a particular content router can request to be registered. Using the information marketplace idea described above, the magazine may seek advertisers or the advertisers may contact a magazine that servers a population they want to reach.

It is also important that a registration protocol tolerate very large latencies between phases of the protocol. Though it is possible to perform the registration process completely automatically, in a commercial setting (or in any situation where content is going to be controlled), it may be necessary to queue up requests for a human to decide on. Therefore, it is critical that the registration operations be permitted to take place one at a time: single session registrations must not be required.

A *register-me* request from an information server to a content router must contain sufficient information for the content router to contact the server for additional information. For example, a Web-based content router may export a form with a registration operation that requires the remote system to supply a URL for future contacts. An Semantic File System-based content router might respond to a special query, say with the attribute `register-me: server-name`. This query would be required to contain attributes for host name, host address, and port number that would allow the content router to contact the registering semantic file system. The content router should return a simple acknowledgment specifying whether the operation was successful. The content router will request further information at a later time. The registering server should not send a content label because the content router may restrict the size of content labels or may not want to devote too much space registration requests. However, this operation may support an optional parameter that specifies the size of the content label the information server would like to export.

A content router may submit a *send-registration-information* request to an information server either unsolicited (to poll for updates or request a new registration) or in

response to a *register-me* request. This operation may contain an identifier that the information server should return with the other information to identify the request to the content router. In addition, this operation must specify any restrictions on the information expected, *e.g.*, limits on the size of content labels. The information server should respond with a simple acknowledgment of the operation's success or failure.

Finally, an information server may post its *registration-information* to a content router. This operation should specify any identifier sent as part of the *send-registration-information* request to which it is a response. The information server should also provide contact information for the forwarding of queries (or user connection referrals) as well as its content label.

2.4 Summary

This chapter presented a content routing system design satisfying the goals of Chapter 1. The design:

- Supports interleaved browsing *and* associative access.
- Supports a uniform query interface that allows progressive discovery of network contents.
- Supports distributed processing.
- Exploits heterogeneity of data sources to organize the search space.
- Propagates descriptions of server contents through a hierarchy of information servers.
- Routes individual queries to servers based on the expected relevance of the server to the query.
- Provides useful suggestions to help users formulate better queries (query refinement).

- addresses the metadata issue by outlining how to build descriptions of server contents.

The architecture scales because the hierarchical structure controls fan out and control over content labels control the resource requirements at content routers. Thus, resource requirements do not grow without bound at any point in the network.

The information loss at each level of the network may inhibit the utility of hierarchies of great height. Exploring this problem in detail is left for future work. However, as Chapter 3 will show, it is possible with current technology to cover all hosts on the present day Internet with a three-level hierarchy. We have tested the performance of our system in small, artificial hierarchies of this height, and performance has been adequate. This suggests that the data loss may not be a debilitating.

Chapter 3

Implementation and Practical Experience

The principle purpose of the implementation effort was to verify the content routing architecture by determining whether such a system would be practical to build, provide acceptable performance, and scale well in a realistic setting. We wanted to understand how to construct content labels, and we hoped to provide a useful service to the Internet community. We constructed four prototype systems using two different underlying protocols to gain practical experience. In particular, the implementations were to answer the following pragmatic questions:

- Can a system help a user cope with the vast amount of information available by providing some organization of the data and automatically suggesting improvements to users' queries?

  Answer: Yes. We have found that content routing systems can be organized in ways that are easy to browse and search and that make sense from a user's point of view. Moreover, we have found query refinement suggestions to be meaningful and useful in day-to-day use. Section 3.3 discusses our implementation of query refinement. The general design was outlined in Section 2.3.2.

- Is it possible to route queries to a meaningful subset of a large number of information servers?

  Answer: Yes. All our implementations have been successful in environments of over 500 servers. See Section 3.2 for some evaluation of the quality of content labels constructed using only simple techniques and limited data.

- Is it possible to define some metadata structures that can be used to organize the information space and provide other assistance to users?

  Answer: Yes. The definition of content labels in Section 2.3.3 together with the practical construction techniques of Section 3.1 shows how this can be done.

- Can the content routing architecture successfully interoperate with pre-existing systems?

  Answer: Yes. The prototypes have successfully interoperated with Wide Area Information Servers (WAIS) [29] and Semantic File Systems (SFS) [21]. Section 3.4 describes a content router implementation built on top of SFS and incorporating WAIS servers. Section 3.5 describes an implementation based on the hypertext transport protocol (HTTP) [4].

- Is there a practical way to extract useful content labels from large numbers of disparate, incompatible servers?

  Answer: Yes. The prototypes successfully build useful content labels from 500 remote servers that operate using completely independent indexing technology and do nothing special to cooperate with their efforts. Section 3.1 describes how our prototypes construct their content labels.

- Will a hierarchical content routing system scale to millions of servers?

  Answer: Perhaps. The performance of the prototypes show clearly that a single content router running on a relatively modest workstation can support 500 remote servers. In fact, it is clear to us that a single content router can support thousands of servers. We have built two and three-level hierarchies, which, given this scale,

would easily cover millions of servers. However, we have not had millions of servers with which to build test hierarchies, and we have no real experience in building content labels for higher level servers. Ongoing research is exploring these issues, but the results so far are encouraging. See Sections 3.8 and 3.5.2 for performance data from the content routing system prototypes.

In order to test a system in a realistic setting, our prototypes exploit the large and growing set of Wide Area Information Servers (WAIS) [29]. The data presented in this chapter come from prototype systems that registered between 492 and 504 WAIS servers during the time of our experiments.

To learn what factors most affected performance and ease of integration into existing environments, we sought to compare two implementation paths using different underlying network protocols. The first implementation strategy was to use the Sun Network File System protocol [66]. This strategy was inspired by our earlier work with the Semantic File System [21]. The second strategy was a prototype based on the HTTP protocol used by the World-Wide Web [4].

Experience with the prototypes has shown content routing to be a promising and practical approach to information discovery in networks of information providers. The prototypes have been very useful in exploring the contents of small networks (consisting of a half dozen servers) and of networks consisting of over 500 servers. Perhaps the most surprising result is that even without detailed data analyses, very simple content labels and query refinement algorithms can form the basis of a useful system. Performance has been largely acceptable (query refinement is often too slow) and the prototypes appear to scale well.

This chapter describes our prototype efforts by detailing how we constructed content labels (Section 3.1), evaluating the content labels (Section 3.2), and describing how we implemented query refinement (Section 3.3). It then gives an overview of implementations built on the Semantic File System protocol (Section 3.4) and on the HTTP protocol (Section 3.5). The chapter concludes with a list of lessons learned from the comparison

of implementation strategies and the current state of our knowledge about content routing (Section 3.6).

3.1 Building Content Labels

It is possible to build meaningful descriptions of servers. Moreover, the prototypes built for this thesis have automatically constructed content labels in two environments: one in which the databases cooperated with statistical analyses of index data (because they were under our control), and another in which information was extracted from hundreds of remote, incompatible servers.

The prototype implementations used two strategies for producing content labels. One approach minimizes content label size and provides semantics like that of a name server. That is, we analyzed index data and chose attributes with small value sets that characterized a server's contents well while providing discriminatory power among servers. These attributes were used in content labels. These content labels indicate which fields a server indexes and provide either a complete set of values for a field, or no values. This arrangement provides a strong semantics: if an attribute does not appear in a content label which lists its field, then it does not appear in any server below that node in the hierarchy, representing a kind of closed world assumption [50]. However, this approach limits what queries can be made while interacting with higher-level content routers. The content labels built in this way were very small, averaging approximately 940 bytes in size.

3.1.1 Content Labels for SFS Servers

Analysis of Index Data

Recall that information providers must balance to competing objectives in the construction of content labels: content labels must represent the data in the collection and export easy to guess to terms, but it is important to chose terms that provide high discrimina-

Server	*# of attributes*	*Server size (MB)*	*Index size (MB)*
comp	564493	65	50.5
rec	309473	43	29.5
users1	247914	184	29.5
nyt	165678	174	29.3
users2	99451	28	15.6
ap	36620	29	3.9
total	1423629	403	158.3
unique	1077522		

Table 3.1 Information servers statistics

tory power. Terms with high discriminatory power distinguish one server from others. It is possible for a content router to filter content labels of the servers it registers retaining only the terms that have some discriminatory power.

Thorough analysis of this information requires access to all the relevant index data. In order to explore how content labels can be generated automatically for large collections of objects, we have gathered attribute statistics on six information servers under our control. (The statistics presented here have appeared in [59, 60, 14]). Table 3.1 gives the following characteristics of the six servers: the number of distinct attributes, the server size (total size of documents in the database in megabytes), and the index size (size of the index data structures). *Comp* is a database we built from the USENET `comp` newsgroup hierarchy, *rec* is the `rec` newsgroup hierarchy, *users1* and *users2* are two indexed user file systems, *nyt* is a database of New York Times articles, and *ap* is a database of Associated Press articles.

To evaluate the discriminatory power of attributes, we gathered data on the distribution of attributes over servers. Figure 3-1 plots the total number of unique attributes versus the number of servers. It shows that `text` comprises a majority of the attributes and that the number of distinct attributes is large. A content label containing all attributes (or even all `text` attributes) might be infeasible. Therefore, one should select only high frequency terms (excluding those on a stop list) or very important `text` terms. See [51, 19] for statistical measures of term significance. On the other hand,

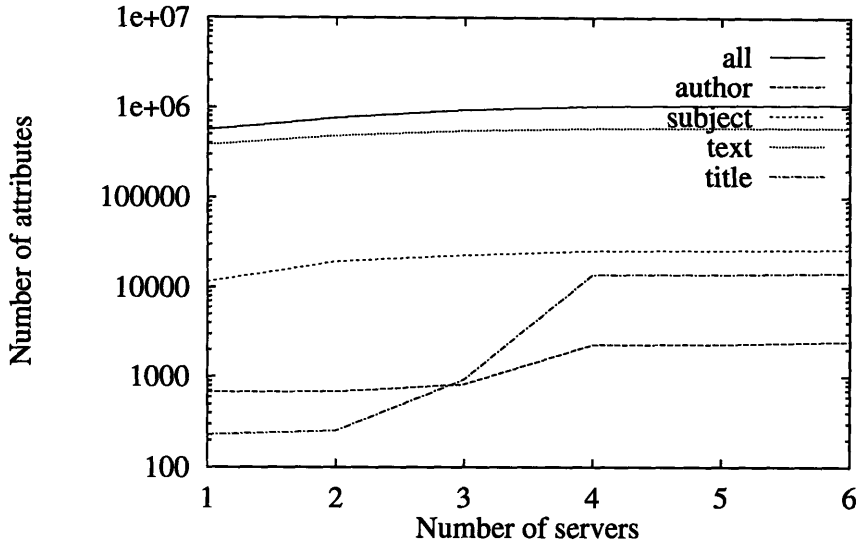


Figure 3-1 Cumulative number of attributes

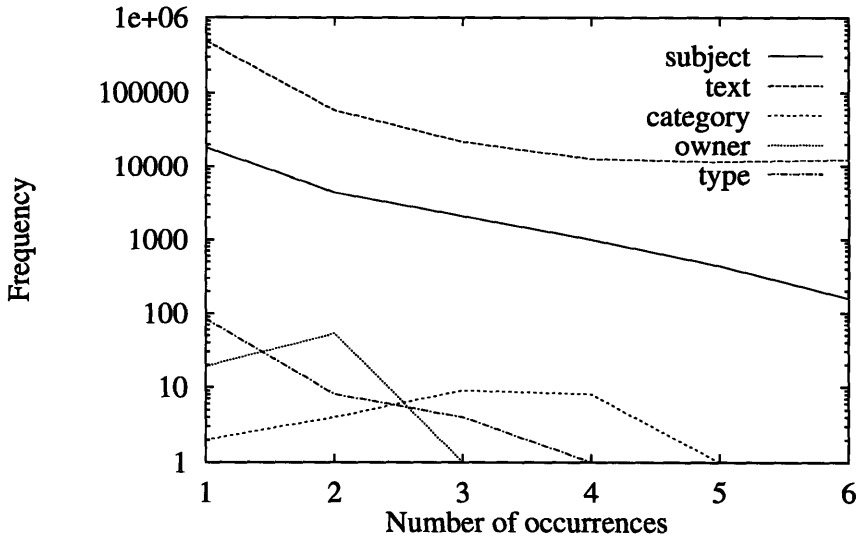


Figure 3-2 Attribute histogram

some information-rich attributes such as `author`, `title` and `subject` have a small set of information-rich values and are thus good candidates to be included in content labels in their entirety.

Figure 3-2 shows a more direct measure of discriminatory power. It shows for a given number of servers how many attributes appear exactly on that many servers. For example, there were 10 `category` attributes that appeared on three servers. If a given attribute has a narrow distribution, that is, it identifies a small number of servers, then it is very useful for routing. Wide distribution terms are useful for describing collections for browsing. As shown in the figure, low frequency attributes such as `owner`, `category`, and `type` have discriminatory power over servers and can be used for content routing. Higher frequency attribute fields like `text` and `subject` are more common so that the most frequent terms can be used for categorizing the collection as a whole and propagated to upper routing layers.

3.1.2 Content Labels for WAIS Servers

Our implementation efforts eventually focused on a second strategy for constructing content labels. We chose to experiment with very simple content labels that consisted of sets of keywords obtained from individual WAIS *source* and *catalog* files. This approach was driven by the desire to provide content routing in a network of 500 Wide Area Information Servers (WAIS). We chose WAIS servers because WAIS was being used more and more for database servers on the Internet. Because our experimental systems involved using 500 WAIS servers over which we have no control, it was not possible to enlist the aid of WAIS administrators to produce value-added attributes. Nor was it possible to do the detailed statistical analyses shown above to choose the content label attributes because we did not have access to the WAIS index structures (and browsing all the documents was not feasible). Therefore, we had to construct all the content labels, even though the content routing system design delegates the responsibility for content label construction to the individual servers.

A WAIS *source* file contains a short description of a server, including contact information (*e.g.*, host name, host address, database name, administrator) as well as a short list of keywords and a natural language summary of the server's contents. In our sample, the median size of the source files was under 800 bytes. While they may enable one to categorize a server into a general domain such as biology, source files are not adequate by themselves for query routing or refinement. Here is the entire WAIS source file for the National Science Foundation awards server:

```
(:source
   :version  3
   :ip-address "128.150.195.40"
   :ip-name "stis.nsf.gov"
   :tcp-port 210
   :database-name "nsf-awards"
   :cost 0.00
   :cost-unit :free
   :maintainer "stisop@stis.nsf.gov"
   :description "Server created with WAIS release 8 b4 on
                 Apr 21 09:01:03 1992 by stisop@stis.nsf.gov

   This WAIS database contains award abstracts for awards made by
   the National Science Foundation.  The database covers from the
   beginning of 1990 to the present (no abstracts are available before
   1990).

   If you use WAIS to retrieve these documents, We'd like to hear about
   your experience.  Please write to stis@nsf.gov.

   You might also be interested in the nsf-pubs database which contains
   NSF publications.  It is also on host stis.nsf.gov."
)
```

On the other hand, WAIS *catalog* files contain considerably more information. A catalog file contains one *headline* for each document in the WAIS database. A headline is typically the subject of a message or the title of an article, though some servers choose less useful headlines such as local file names. Thus, though a catalog file is much smaller than the WAIS index, it often contains information-rich terms that characterize the database fairly well. Here is the beginning of the catalog file from the National Science Foundation awards WAIS server:

```
Catalog for database: ./nsf-awards
Date: Sep 23 07:21:33 1994
63920 total documents

Document # 1
Headline: Title: Summer Undergraduate Research Experience Fellowships in the
DocID: 0 2049 /home/pub_gopher/.index/ftp/awards93/awd9322/a9322138

Document # 2
Headline: Title: NYI: Dedicated VLSI Digital Signal and Image Processors
DocID: 0 2682 /home/pub_gopher/.index/ftp/awards92/awd9258/a9258670

Document # 3
Headline: Title: Prediction of Soil Liquefaction in Centrifuge Model Tests
DocID: 0 2681 /home/pub_gopher/.index/ftp/awards91/awd9120/a9120215

Document # 4
Headline: Title: Igneous-related Metallogenesis of the Great Basin
DocID: 0 707 /home/pub_gopher/.index/ftp/awards90/awd9096/a9096294
.
.
.
```

Recall that a *retrieve* operation on a collection document is defined to return a human readable form of the collection's content label. Figure 2-3 showed how one of our prototypes simply displayed the concatenation of a WAIS source and catalog file as the content label when a user retrieved the collection. In response to a retrieval request on a collection, our latest prototypes return a list of the indexed terms from the collection's content label. The terms are sorted by document frequency. This gives the user a good characterization of the contents of a collection. Here is the start of a sample response to a *retrieve* on the **nsf-awards** collection which lists proposals and awards for research by the National Science foundation. Each term in the content label is preceded by its document frequency:

```
text:
4916 mathematical
4778 sciences
2520 studies
2453 science
2381 award
2081 presidential
```

```
1795 cooperative
1699 development
1619 laboratory
1565 collaborative
1399 study
1355 engineering
1280 physics
1277 molecular
1242 program
1209 systems
1193 dissertation
1170 structure
1136 dynamics
1076 conference
1072 chemistry
1008 computer
```

We started with the list of source files available from the WAIS directory of servers. These 504 source files occupy approximately 750KB. As stated above, the median size of the source files was just 800 bytes.

Most WAIS servers return a catalog file automatically in response to an empty query. We therefore submitted an empty query to all of the WAIS servers described in the 504 source files. The 386 catalog files retrieved in this way occupy 191.1MB. Some of the catalog files, such as those with article titles, are very useful. Others, such as those in which headlines are file path names, are not useful. The largest catalog file occupies 16.7MB and contains 113,223 headlines; the smallest one is 79 bytes without any headlines.

Logically, a content label for a WAIS server is the concatenation of the server's source and catalog files. For query routing, we only need to store the set of terms in the document headlines (with duplicates removed). As a result, the total size of the query routing database is only 15.4MB. This information is kept in a local WAIS database. (See Section 3.3 for information on query refinement.)

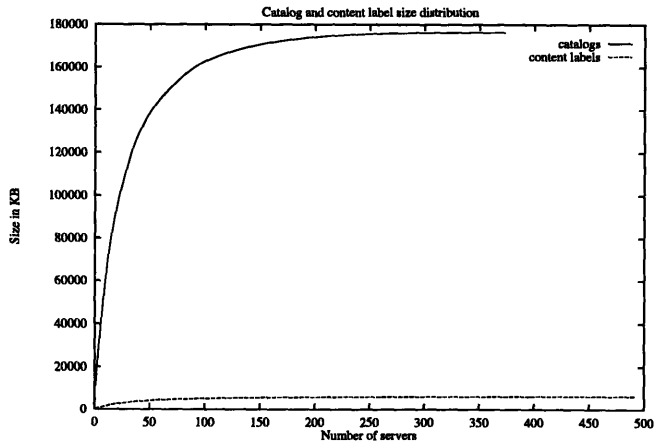


Figure 3-3 Catalog and content label size distribution

term	source files	content labels	exhaustive search
carcinoma	0	5	34
discovery	6	23	223
multimedia	9	45	189
video	6	62	273

Table 3.2 Number of relevant servers

3.2 Evaluating Content Labels

It is possible to route queries to meaningful subsets of a large number of servers. To evaluate how well the prototype routed queries based on the automatically generated content labels from WAIS servers, we collected some statistics on the results of search using independent information about the contents of WAIS servers. Table 3.2 shows how many servers were found using source files, our prototype's content labels, and a brute force search of all servers looking for documents containing the specified term. The server recall capability of content routing depends on the definition of a relevant server. If a relevant server is defined by terms in document headlines or in WAIS source files, then our prototype server has 100% recall by definition. If server recall is defined by terms in document text, then in Table 3.2, server recall varies between 10% and 22%. While this recall rate is modest, it is clear that it could be substantially improved with enhanced WAIS server support for content label generation. Nonetheless, these results are comparable to similar results for GlOSS [25], and they were obtained with no cooperation from remote servers or sophisticated processing of index data. Thus, while the prototypes find a 'meaningful' set of relevant servers, there is much room for improvement. I feel that a more complete system would do at least as well as existing systems — certainly if the prototypes had all the same data, they could do as well. On going research is focusing on the use of Web robots to gather data, and this should allow a more direct comparison of content routing to existing global indexing strategies.

One could argue that the prototype finds servers that are more relevant than others because the terms obtained from headlines are more important than terms found in

a document body. In a subsequent experimental trial using the term `carcinoma`, our system found 19% of the relevant servers using content labels. However, these servers provided 40% of all relevant documents. It is certainly the case that a better statistical analysis of a collection would yield better content labels and better recall, and current work is focusing on addressing this issue.

3.3 Implementing Query Refinement

It is possible to provide the user with meaningful suggestions for the formulation of better queries. This is the purpose of query refinement.

Query refinement has proven to be a crucial component of a large distributed information system. The system must share its knowledge with the user to help formulate queries that reflect the user's interests and can be satisfied within reasonable performance constraints.

As seen in Chapter 2, a user of a content routing system may list the fields available for querying as well as the values indexed for any given field. These operations are implemented as straightforward traversals of the index data structures and are no different than their counterparts in the Semantic File System implementation. Of course, since WAIS systems do not support attribute-based queries, these operations are of less significance in our later prototypes.

The query refinement feature of the current prototypes uses the conditional probabilities of term collocation to recommend terms to the user. (Section 2.3.2 describes how conditional probabilities are used.) Tables 3.3 and 3.4 show the terms recommended by one of our prototypes for certain single term queries. Notice that many suggested terms have a strong semantic relationship with the query term. Notice also that terms sometimes include misspelled terms, which points out the need for a query semantics that does not include exact matching. Extending the query semantics remains for future work.

buddhism	distributed	infosystems	multicast
electronic	systems	comp	routing
buddhist	computing	gopher	switch
coombspapers	system	www	communication
archives	algorithms	available	txt
ftp	parallel	gis	ietf
papers	control	ncsa	switching
net	processing	mosaic	protocol
doc	proceedings	server	packet
zen	simulation	wais	internet
poetry	memory	systems	iinren
tibetan	algorithm	release	atm
ritual	data	version	systems
theravada	operating	msen	ethernet
wustl	database	marca	enabling
wuarchive	computer	emv	drafts
otherwork	comp	client	draft
mirrors	networks	john	connections
edu	research	user	vax
asia	libtr	uigis	unrecognized
zenrmat	based	math	novell
western	programming	info	networks
research	parameter	geographic	network
discussion	design	ftp	mahmood
buddha	analysis	boo	distributed
vietnamese	podc	beta	architecture
students	performance	announcing	algorithms
journal	detection	announcement	udp
asian	applications	list	trdln
yasutani	network	jonzy	todd
univ	fault	information	sun
ulkyvm	time	unix	sgi
thes	termination	ubvm	mckinley
theology	optimal	rfd	icpp
thai	model	released	bgp
teaching	communication	lindner	atomic
taoism	programs	interfaces	support
	efficient		sparc

Table 3.3 Query refinement examples from 6 April 1995

multimedia	poland	video	wireless
multi	research	indigo	psi
media	fund	systems	service
sun	list	games	data
mail	joint	edu	support
comp	data	wustl	psilink
discussion	mailing	wuarchive	enhances
computer	polish	sun	adds
bitnet	ussr	rec	networks
interactive	privacy	sgi	originally
mmedia	cooperative	board	electricity
environment	climatic	sparc	martin
package	western	ftp	termin
list	monthly	spencer	marc
instruction	edu	live	internet
demo	discussion	indy	horow
sgi	physics	digital	communication
sciences	ubvm	help	wpi
video	msc	framer	systems
rom	devoted	usenet	src
information	culture	output	mouse
zoology	cam	ntsc	mike
windows	bitnet	interactive	mail
system	warsaw	system	lord
platforms	setting	reverse	local
mmm	reports	mac	list
biological	observations	info	laboratory
teaching	nfs	graphics	jipping
laboratory	jmr	bit	information
education	international	conferencing	dcom
data	help	frame	cross
based	held	card	comp
vmtecmex	factboot	software	commercial
technologies	development	options	brown
rare	cpsr	network	available
	country	cable	alan
	cia	boards	worl
	nodak	recording	weekly
	news		weber

Table 3.4 More query refinement examples from 6 April 1995

term	number of headlines	processing time
buddhism	82	7.2s
distributed	324	10.4s
infosystems	87	6.2s
multicast	126	7.1s
multimedia	320	12.8s
poland	123	11.1s
video	349	11.0s
wireless	48	4.9s

Table 3.5 Query refinement performance

Query refinement is a more complicated operation that requires some auxiliary data structures of its own. Our prototype condenses the information in the WAIS catalog files by removing file names and document identifiers and other extraneous information. This reduces the 191.1MB of catalog file data to 62MB for 504 WAIS servers. The prototype maintains a local WAIS database, separate from the routing database, of all headlines indexed as one-line entries. The resulting indices are 196MB.

Given a refinement request for a query, the refinement database is used to identify the headlines from servers in the current result set that satisfy the query. Then the terms from the headlines that do not appear in the query are sorted by frequency, and the 40 most frequent terms are presented to the user. Thus, the prototype implements a ranking based on conditional probability of term collocation. This corresponds to the identify ranking or $\mathcal{M}_1$ in Section 2.3.2. As shown in Section 2.3.2, our experience has been that our rankings are similar to that of an entropy function because conditional probabilities from our headline data quickly become very small.

We have also tested the performance of the query refinement feature. Table 3.5 gives performance data for refinement of single term queries. The table gives the number of document headlines containing the term in the query and the wall clock elapsed time for the operation. The figures show that, even with our first naive implementation of query refinement, performance is adequate for interactive use.

It is clear that production systems will require a much more efficient implementation

of query refinement, both in time and space. Thus far, our implementations have been very naive because we focused on how quickly we could do the implementation. Our implementations use off-the-shelf indexing software to build databases that map keywords (as strings) to document names. There is a lot of string processing in the implementation. It is clear that we could reap considerable performance gains by hashing terms and using the hash values in the index. Hash values could be locations in the B-tree, or some other perfect hash function. (See [67, Section 13.5] for a discussion of perfect hash functions and how to compute them.) A perfect hash function is easy to compute because the system has all the terms in advance, and much processing can be done off line. The headlines could then be stored as lists of hash values corresponding to the terms in the headline. An inverted index of this file (a B-tree, for example) would map the integer hash value of a term to the set of byte-offsets in the headline file that correspond to headlines containing that term. Set union and intersection operations on the resulting lists of integers would be used for determining what headlines match the query, and for manipulating the lists of collocated terms. I would expect an order of magnitude performance improvement with these techniques.

3.4 SFS-based implementation

This section describes the general structure of the SFS-based content router prototypes, the SFS to WAIS gateway that that enabled the incorporation of WAIS servers into the information hierarchy, and the performance of the prototype. These prototypes showed successful interoperation with both SFS-based information servers and WAIS.

3.4.1 Structure of the SFS-based prototypes

Our initial prototype implementations of the content routing system provided query routing to an extensible number of servers via the Semantic File System interface. This interface was used for both information servers and content routers. Experimenting

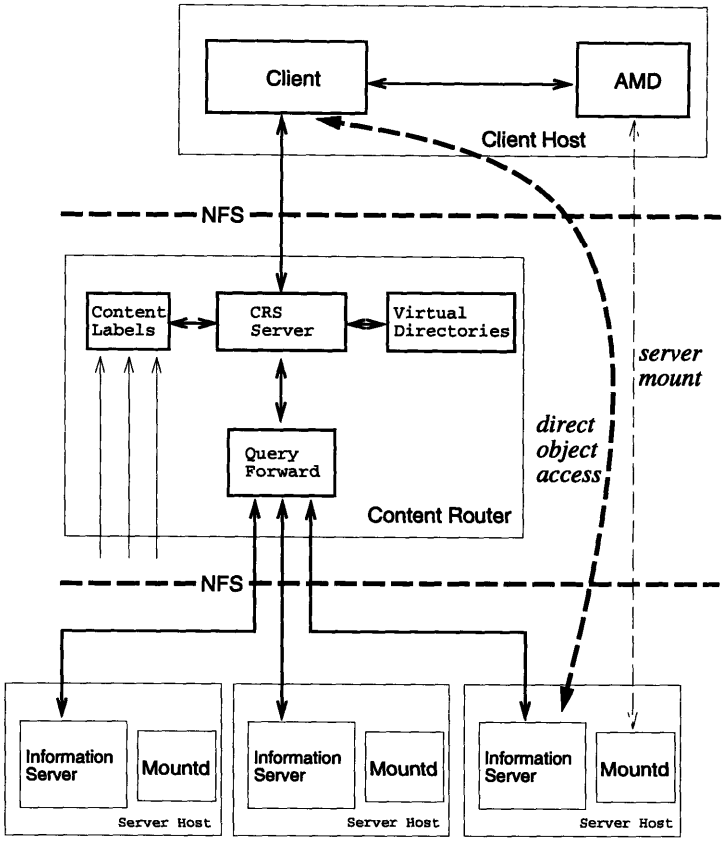


Figure 3-4 A prototype content router based on SFS.

with the Semantic File System interface had the advantage of building on previous work within the group. It was a simple matter to construct a prototype with several databases (including New York Times and Associated Press news wire services as well as user file systems). These prototypes supported only conjunctive queries. An SFS-WAIS gateway enabled the construction of a system that routed queries to hundreds of servers. Because it is implemented as an SFS server, our content router is a user level NFS server that may be mounted by any NFS client. Figure 3-4 shows the architecture of our prototype content router implementation.

Path names that pass through the NFS interface to the server are interpreted as queries and results are returned in dynamically created *virtual directories.* The server computes the contents of virtual directories only on demand, *i.e.*, only when the user performs a `readdir` NFS operation. Queries at the content router apply to the content

labels for the collections registered there.

The server performs an expand operation by forwarding the query (and subsequent refined queries) to the set of servers whose content labels match the query. The merged results are presented to the user. Our SFS-based implementation of the search operation used syntactic hints embedded in queries. These hints indicate to the router which terms should apply to content labels and which should be forwarded to information servers.

The content router interposes itself between a client and an information server during query processing when results from more than one server must be combined. This approach is called *mediated processing* because the content router mediates the responses from multiple information servers to present a collective result.

When only a single server is sent a query or once a document of interest is identified, a client is instructed to bypass the content router and contact the end-point server directly. This approach is called *direct processing*. Direct processing allows a client to obtain maximum performance from the end-point server and also minimizes content router load. In the SFS-based prototypes, client to server forwarding is implemented by having the content router return a symbolic link to a client in response to a request for a virtual directory or file. The symbolic link refers to the end-point server and is dynamically resolved by the client with the help of an automount daemon [45].

This approach required the clients to run a version of the automount daemon modified to perform mounts on arbitrary ports. The clients also required two mount maps. The mount maps were general purpose and did not require any information about servers. Thus the client requirements were very small and did not require updating. The prototype used a special directory name to indicate that the server should mediate all requests and not return links to be interpreted by the automount daemon. In this case, the clients required no special code whatsoever, they merely mounted the content router and performed directory operations. (See Appendix A for these mount maps and other details about the SFS-based implementations.)

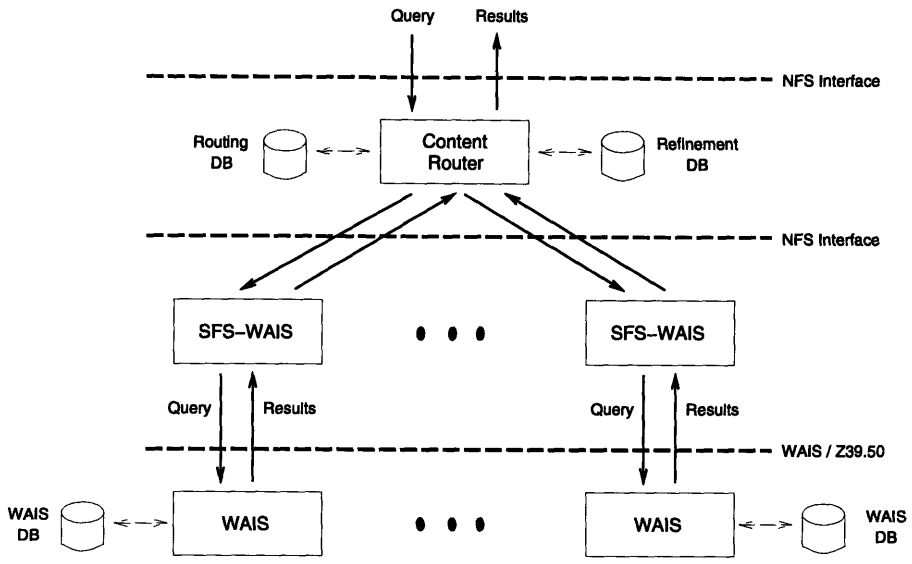


Figure 3-5 Structure of the content router with SFS-WAIS gateways.

3.4.2 The SFS-WAIS Gateway

To provide access to WAIS servers, which do not support the SFS interface, we implemented an SFS-WAIS gateway [14]. The SFS-WAIS gateway translates SFS queries into WAIS questions and uses the public domain client WAIS code for querying WAIS servers. The gateway can be used directly like any semantic file system. WAIS uses a version of the Z39.50 protocol for network-based information querying and retrieval [39]. Figure 3-5 shows the prototype content routing system together with the SFS-WAIS gateways.

The content router and SFS-WAIS gateways are implemented as separate processes (all NFS user-level servers) that can be run on the same or different machines. The client mounts the content router as a standard NFS volume. The SFS-WAIS gateways are mounted on the client and on the machine where the content router executes (possibly via the automount daemon). As before, the content router consults its routing database to determine relevant servers. Then, path name queries containing the identification of relevant servers are forwarded to SFS-WAIS processes. When all results are obtained, they are merged and returned to the client.

The result documents appear to the client as symbolic links to files. When a document

file is accessed, the **readlink** NFS call is intercepted and the document is retrieved from the WAIS server. The document's file name is constructed from the headline of the document obtained as a result of the query. Here is an example of querying the **zipcodes** WAIS server via the gateway:

```
=> Select  source:zipcodes and text:warsaw
=> List-results
14569_Warsaw__NY_.1     22572_Warsaw__VA_.2
28398_Warsaw__NC_.3     41095_Warsaw__KY_.4
43844_Warsaw__OH_.5     46580_Warsaw__IN_.6
55087_Warsaw__MN_.7     62379_Warsaw__IL_.8
65355_Warsaw__MO_.9
```

The content routing system operations, like **select** are translated into ordinary file system commands, like cd with the arguments reformatted into SFS path names.

Conjunctive queries are of special importance because they narrow the search space. However, since many WAIS servers do not implement conjunction, our gateway implements the boolean **AND** operator by running a WAIS query for each query term and calculating the intersection of the result sets. Although it is computationally expensive (the server must find several, possibly large, result sets), it was the only way to use all existing WAIS servers.

3.4.3 Performance of the SFS-based prototypes

We have used a variety of configurations of our content routing system. We measured the performance of query routing in a small system of four semantic file systems, and we measured the performance of a single query router providing access to 492 WAIS servers located around the world. We also verified that direct processing achieves better performance than mediated processing in a multiple-layered system.

Table 3.6 shows representative performance of a content router. We expect that we will be able to further lower query processing times on our system because our prototype implementation is not tuned. The information servers run on an SGI 4D/320S and a

Example query	*Number of servers*	*Number of docs*	*Search time (sequential)*
`library:users and owner:`	1	88	1.2s
`library:users and text:semantic and owner:sheldon`	1	22	0.6s
`library:users and text:semantic and text:library and owner:sheldon`	1	9	1.2s
`location:mit and extension:video`	2	60	2.6s
`location:mit and owner:gifford`	2	31	1.4s
`text:toys and text:williams`	4	27	9.9s

Table 3.6 Routed query performance on four local SFS servers

Example query	*Servers*	*Docs*	*Down servers*	*Parallel search*	*Seq. search*
`buddhism and tibetan`	5	71	0	396.0s	752.2s
`distributed and synchronization`	6	23	3	674.6s	1442.1s
`infosystems`	9	317	0	63.2s	117.0s
`infosystems and gopher`	9	132	1	680.5s	743.7s
`multicast and broadcast`	15	32	8	79.3s	379.3s
`multimedia and authoring`	12	47	1	108.8s	196.9s
`poland and culture`	11	11	0	47.6s	494.7s
`video and quicktime`	12	75	4	106.0s	849.1s
`wireless and mobile`	6	6	0	48.6s	110.9s

Table 3.7 Example query performance routing to 492 WAIS servers

heavily loaded DEC Microvax 3500. The content routers and clients were run on Sparc Station IPX's. All these machines were interconnected with a 10Mbit/s Ethernet. These tests did not process queries on multiple servers in parallel.

We also used our prototype to locate and access documents on a collection of 492 WAIS servers, and we gathered statistics on some example queries. The content router in this case ran on an SGI 4D/320S, the SFS-WAIS gateways ran on a Sun SparcStation 10, and the WAIS servers were distributed throughout the Internet and ran on unknown types of machines. Recall that since WAIS servers do not in general support conjunction, our SFS-WAIS gateway is forced to run a separate remote query for each term and compute result set intersections locally. This greatly increases processing times and encourages

users to restrict themselves to highly specialized query terms. The statistics are presented in Table 3.7 for simple conjunctions of terms. We report the number of relevant servers as determined by content labels at the content router, the number of documents found, the number of unreachable servers and the search latency for each query. The table compares our content router conducting searches in parallel with a sequential search using `waisq`, a program in the WAIS distribution. In general, parallel searching performs better than sequential. However, the speedup is well below linear: the latencies on different servers vary so much that the parallel searching time is strongly limited by the slowest server. Some servers contain large databases, are accessed by many users, and have limited processing power. These servers limit the performance of any search. For example, the long search time for `infosystems` and `gopher` was due almost entirely to the processing time at one server (`biosci`) that continued to run long after other servers had finished. Moreover, one can experience 100% changes in search latency depending on whether the search is performed during the day or at night. Even if we take into account the delays at overloaded servers, the reported search times are long. However, we should bear in mind that the principle alternative to using our content router is an exhaustive search of all servers, an operation which takes several hours to complete.

While a hierarchical system facilitates locating objects, it introduces communication delays and performance restrictions that become intolerable for larger sets of applications as the layering gets deeper. In particular, server load is proportional to the resources consumed by clients accessing the server. It is important for the performance of the system, and thus of client applications, to minimize the resources consumed by clients. Therefore, once a client identifies an object, or once the content router has discovered that there is only one information server appropriate to a client's request, the system must provide direct access to the end-point server. The client then does not incur the latency of a multi-layered communications path, and the system does not have to provide resources that are not necessary to serving the client's request.

We performed a suite of experiments with content routers and four SFS servers to

test the performance implications of mediated versus direct access to objects. Figure 3-6 illustrates the performance penalty incurred when content routers mediate file read requests in our prototype. The figure plots throughput achieved at the client in Kbytes per second versus load at the content router. For our experiment, the load was generated by running 0, 1, or 2 processes that continuously generated lookup and read requests through the content router. Queries always go through the intermediate content routers. In one set of trials, all client requests were mediated by the content router, including file reads. For this set, the throughput for the case where the content router is not used is presented for comparison (above the label "direct"). In the other set of trials, the content routers provided the client direct access to the server via symbolic links interpreted by the client's automount daemon. As expected, clients with direct access to end-point information servers achieved better performance, especially as the load at the content router increased.

The content routers provide unmediated access to data by returning symbolic links. These links are interpreted by a version of the Berkeley automount daemon [45]. Mediated accesses naturally go through the content router. In Figure 3-6, each data point represents the average of four trials each of which reads a 35 MByte digital video data file. The information server was running on an SGI 4D/320S. The content routers and clients were run on Sparc Station IPX's. All these machines were interconnected with a 10Mbit/s Ethernet. These figures compare with an average throughput of 830 KByte/s for our standard NFS file access.

Figure 3-8 shows system throughput from a standard NFS client (on a Sun SparcStation IPX) in the following configurations:

- *NFS* — direct NFS service to our group file server.
- *SFS* — via the SFS server on our file server.
- *CRS-1* — via a non-mediating content router on the client machine.
- *CRS-2* — via a mediating content router on the client machine

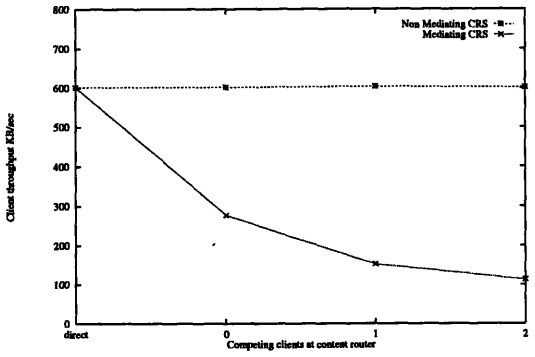


Figure 3-6 Throughput at a single node versus load

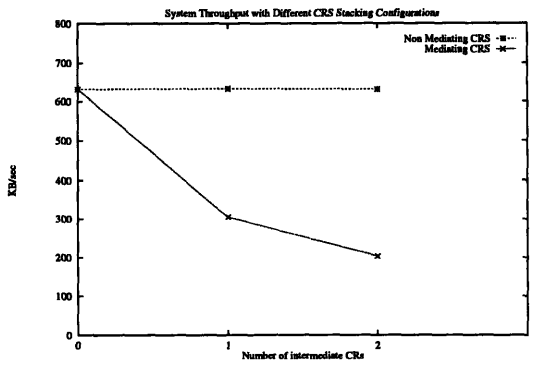


Figure 3-7 Throughput versus mediating content routers

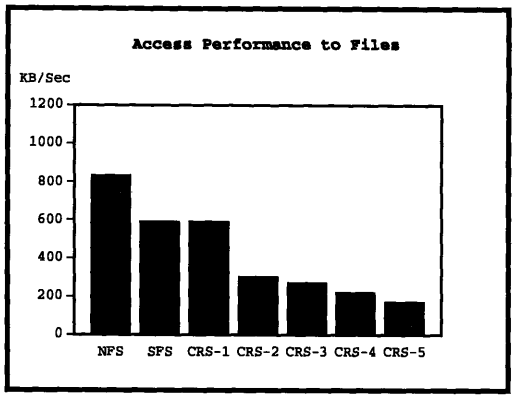


Figure 3-8 System throughput to files for different CRS configurations

- *CRS-3* — via a non-mediating content router that is running through a mediating content router both on the client machine.
- *CRS-4* — via two mediating content routers running on the client machine.
- *CRS-5* — via two mediating content routers, one running on the client machine, and the other running on another machine on the local net.

Mediated access has such a dramatic impact even for low loads because our implementation is not multi-threaded and does not do read-ahead. That is, it is not an optimized file server. Nonetheless, the impact of content router load is inherent in any implementation that forces clients to use mediated data access because resources at each server are finite. The more layers there are in the network, the more resources are consumed by

each client, and the greater the likelihood that a client will suffer query router induced performance degradation.

The appearance of efficient network searching servers like the WAIS content router may aggravate the load problems of some popular WAIS servers. As the number of users grows, the load on the servers will also increase. It thus will become more important to investigate solutions to this problem such as caching and replication.

Availability of servers is also a problem. As can be seen from the table, at least one server was unreachable for half of the sample queries. In fact, any large network of servers will nearly always have some servers down or unavailable. Of course, servers will ultimately want to be highly available, and will use replication techniques to achieve this aim. However, the prototype content router required some mechanism for dealing with unavailable servers. Our initial solution has been to create virtual documents whose names signal the user that some server(s) did not respond (e.g. `comp.sys.sgi.misc.src:unreachable`). A better policy would be to keep track of unavailable servers, check periodically if they are up and eventually rerun a query.

Content routing based on WAIS catalog files has resulted in some unexpected behavior. For example, the query from `text:buddhism and text:tibetan` was run on 5 servers and documents were found only on 4 servers (all servers were operational). A closer look at the catalog file of the missing server explained the problem. There is a headline with a path name containing terms `buddhism and tibetan`, however the search on the WAIS server produced no result, because the file itself does not contain the terms:

```
ftp.uu.net:doc/papers/coombspapers/otherarchives/electronic-buddhist-archives/
        buddhism-tibetan/
```

Our experience with using the content router for locating and accessing WAIS servers shows that supporting boolean operations on servers is crucial to efficient searching. Only specialized terms can be efficiently used in our prototype, because general terms involve long searches with many result documents, even if a conjunction of two general terms returns a small result set.

Our use of an NFS-based protocol has advantages and disadvantages. Because NFS is widely understood, and because our servers use NFS at both ends, we are able to reconfigure our system and use new machines easily. NFS also makes it very easy for clients to participate, and allows us to leverage existing code. One can run editors, compilers, and standard file system tools, all without modification.

Unfortunately, we still found NFS to be a poor protocol on which to build a content router. In retrospect, it is clear that content routing, especially to large numbers of widely distributed servers of varying performance, will entail large and varying latencies. NFS systems do not tolerate large latencies or failures well as a rule, and error reporting and recovery are awkward in the protocol. We frequently had difficulties with client code on both our hardware platforms (both in the operating system and the automount daemon) blocking. NFS operations are also difficult or impossible to abort from clients. Ultimately, a more latency tolerant protocol, based on TCP for example, is required. The content routing architecture, however, is independent of the underlying communication protocol. We chose to use HTTP for subsequent prototypes (see below), but we could also use Gopher [1], Z39.50 [39], or any other mechanism that provides distributed access to servers.

3.5 HTTP-based implementations

This section presents the HTTP content routers for WAIS servers [58]. Again, the goal was to exploit the large information space contained in available WAIS servers and to see whether HTTP would provide a better underlying protocol for a content router. We also hoped to provide a useful service to the WWW community.

Once again, the prototypes successfully interoperate with WAIS servers. The most recent prototype also supports annotated links to HTML documents, and will readily support the integration of World Wide Web servers. Support of other server types is left for future work.

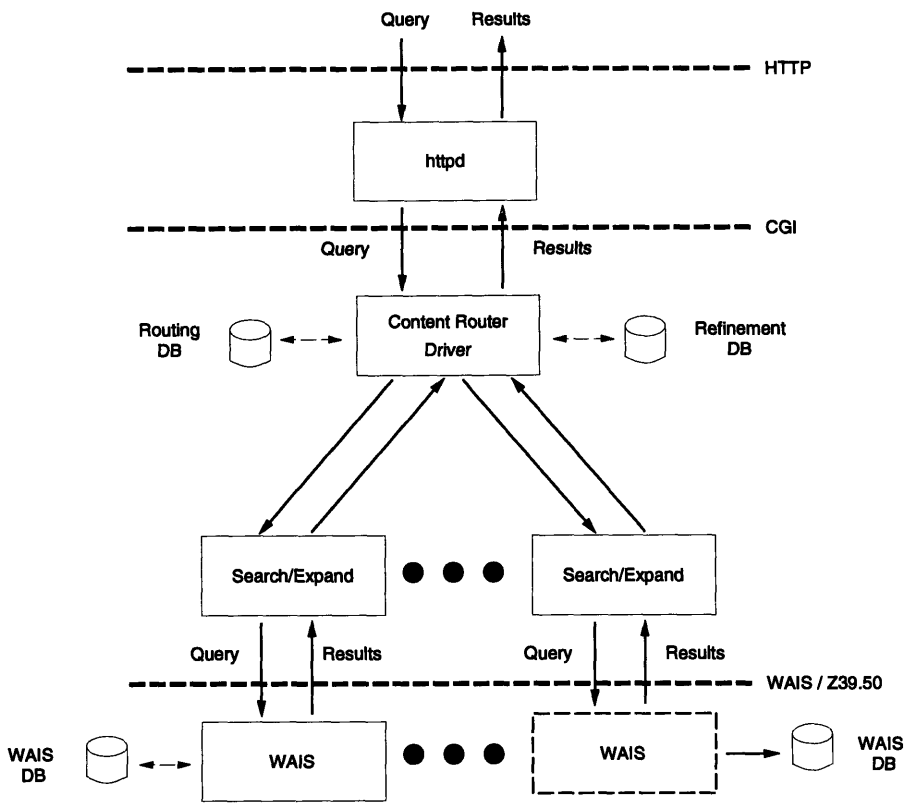


Figure 3-9 Structure of the HTTP-based content router.

The remainder of this section describes the structure of the World-Wide Web content router for WAIS, and reports on our experience with the system and its performance.

3.5.1 Structure of an HTTP Content Router for WAIS

The implementation is based on the framework provided by the HTTP World Wide Web protocol. Since HTML documents and HTML Forms provide the most widely used multimedia display format in use today [6], we chose to use HTML Forms for the system's input and output. Figure 3-9 illustrates the structure of or implementation.

The prototype interacts with the user by supplying HTML forms that can be retrieved and filled in using browsers such as Mosaic[1] and Netscape[2]. These documents can also be

[1] `http://www.ncsa.uiuc.edu/SDG/Software/Mosaic/Docs/help-about.html`
[2] `http://home.mcom.com/home/faq.html`

retrieved by an automatic process, which is how query forwarding is done in a hierarchy of servers. When the client has filled out and submitted a form, the HTTP daemon (`httpd`) running on our server invokes the prototype's driver, which is implemented as a CGI POST script. The prototype interprets the operation and parameters entered by the client, computes a response, and constructs another HTML document with the results. Typically, the resulting document is itself a form that allows the client to choose new parameters and operations and also provides links to remote servers and documents.

For example, Figure 3-10 shows a form that resulted from a user's **expand** request for the query `text: chromosome`. The content router found three relevant servers. The name of each server is a link that will connect the client directly to the named WAIS server. (The URL for the last WAIS server is visible at the bottom of the figure because the mouse is positioned over `fly-amero`. After each server name, the phrase `content label` appears in parentheses. Each of these is a text anchor that, when selected, will display a human readable version of the appropriate content label. The user may chose to perform additional operations, including an exhaustive search, by clicking on the appropriate diamonds at the top of the form, typing in additional query terms (if necessary), and clicking on the submit button.

Links to remote servers by-pass the content router to allow direct processing. In order to make use of links to WAIS servers and documents, the client browser must use a version of the World-Wide Web library routines compiled with the freeWAIS library. Our newer HTTP-based prototype accepts a parameter that requests that the content router return links back through the content router rather than directly to WAIS servers. This form of mediated processing supports clients that cannot interpret the WAIS links.

The prototype implements query routing and refinement using local WAIS databases accessed via WAIS library routines. The content router spawns parallel processes to search remote WAIS servers concurrently. The router and the parallel search processes use the WWW library to search remote databases. The WWW library routines are compiled with the freeWAIS library, and are thus able to contact the remote WAIS

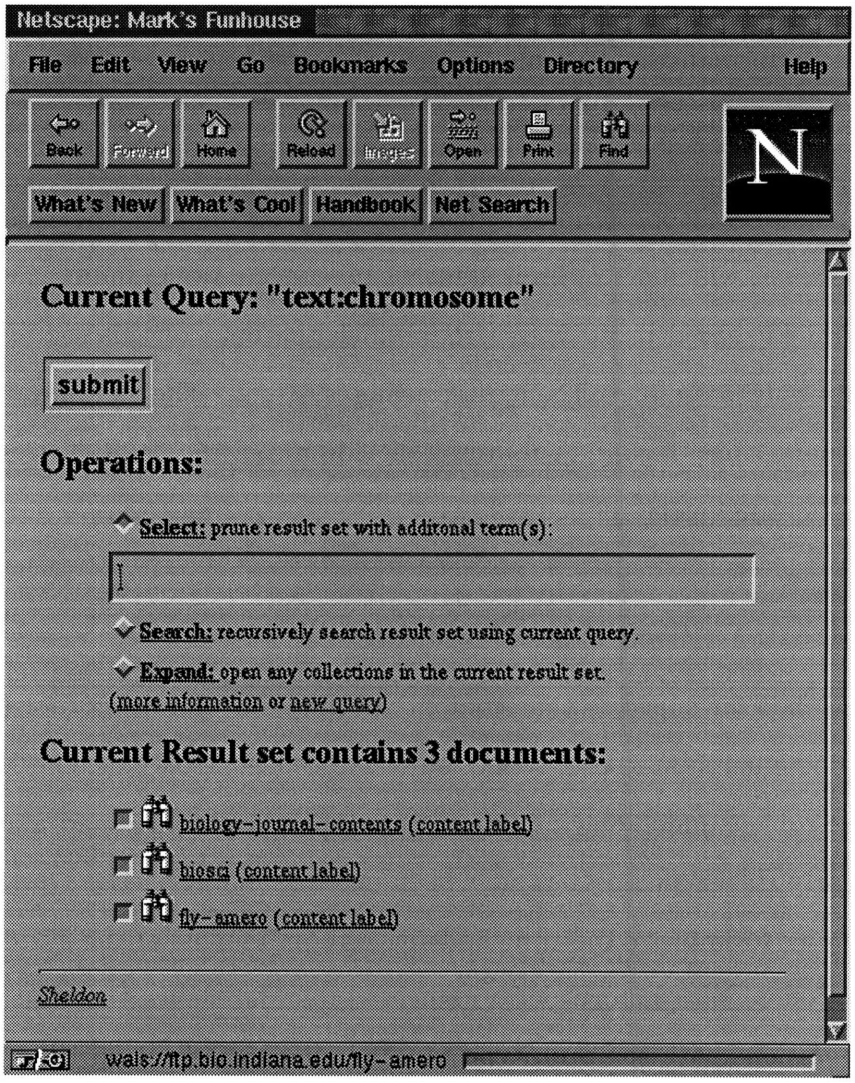


Figure 3-10 A sample HTML form

Operation	Query	Time (min:sec)
expand	`communication`	0:05.07
refine	`communication`	5:35.51
refine	`communication and networks`	1:06.51
search	`communication and networks and routing`	1:29.67
retrieve	`"This Week's TechLink"`	0:09.27

Table 3.8 Performance on sample queries

servers.

The current prototypes implement all the operations in table 2.1 except for *show-fields* and *show-values*, which are of limited utility given that WAIS indices are keyword based. The first HTTP-based prototypes (from which the data of this section were collected) implements only a two-level content routing system. The newer prototype supports arbitrary hierarchies of content routers that communicate via HTML forms, but is only now implementing query refinement.

3.5.2 Experience and Performance

Experience thus far suggests that content routing is a promising tool for content-based access to documents in a large collection of information servers. The HTTP protocol is much more robust under varying latencies and handles failures and aborts smoothly. In fact, the early HTTP prototypes have been made available on the World-Wide Web. We are continuing to improve the time and space requirements of the system, but even now the system offers satisfactory performance. Table 3.8 shows the time to execute some sample operations with the prototype running on a SparcStation IPX.

The *expand* operation was applied to a collection consisting of all the remote WAIS collections. This operation consulted the local routing database but did not entail any remote operations. The *refine* operations consulted the local refinement database. The second *refine* operation was much faster than the first because the document space had been effectively reduced by the previous *refine* and *select* operations. The `refine` operation is usable, but slow. As stated in Section 3.3, we are still improving the query

refinement process, and there are many obvious ways to tune the current, rather naive implementation. Also, based on experience with a prior prototype, we expect a factor of 3–5 speedup when we move the implementation to a SparcStation 10. The *search* operation involved searching 13 remote WAIS servers scattered around the world and merging the results. During the *retrieve* operation, the client contacts the remote WAIS server directly rather than going through the content router. Thus, retrieving this 132K byte document did not involve our server at all.

We have found the power of boolean queries to be very useful. However, as in our previous prototypes, the content router must implement conjunction for WAIS servers by invoking a separate query for each conjunct and performing a local set intersection operation. This entails a large performance penalty for queries that use general terms that occur in many documents, even if the result set for the query as a whole is small.

3.6 Summary

The four prototypes built as part of this thesis have shown that it is possible to implement the architecture of Chapter 2. They also provide answers to the pragmatic questions set out on pages 7 and 79: It is possible to do automatic query routing to a meaningful subset of a large number of servers. It is possible to give useful suggestions to help a user formulate better queries. There are practical ways to extract useful content labels from hundreds of disparate, incompatible servers. Content routing systems can interoperate with pre-existing systems. Finally, it appears that the system may scale.

We are optimistic about the scaling properties of the system for several reasons. The rudimentary prototypes described here have adequate performance, even without extensive tuning. Supporting 500 servers is not a serious drain on our resources, and it is clear that a single content router can support thousands of servers. Only two content routing layers are required then to cover millions of servers, and we have demonstrated adequate performance in hierarchies of this height, though it remains for future work

to actually incorporate millions of servers into a single hierarchy. The principle issue in scaling is the information loss as the hierarchy grows in height, but at two or three levels, the problem should not be too large. The main problem as the networks become deeper is in representing query refinement information. We are working on clustering techniques that will enable more compact collocation data in content labels that provide a similar quality of query refinement service.

Demanding applications like digital video can achieve adequate throughput because content routers do not mediate object accesses. Experience using a content router for locating and accessing WAIS servers shows that supporting boolean operations on servers is crucial to efficient searching. We were pleasantly surprised at the efficacy of content routing based on such simple content labels. With virtually no tuning based on high-level knowledge or sophisticated statistics, it was not difficult to explore a large set of information servers. We would like better recall performance, but we are optimistic that greater information about a collection's contents will enable us to produce content labels with better behavior. We found query refinement to be essential. We found that content routing requires an underlying protocol that tolerates widely varying latencies and that allows efficient aborts of operations after they have started. Performance of the system is acceptable, though there is considerable latitude for improvements in response time and in the space requirements of query refinement. These improvements are the subject of ongoing research.

Chapter 4

Conclusion and Future Work

4.1 Directions for Future Research

Much work remains to be done in the area of information discovery and retrieval in very large scale systems. There are many legal and economic issues that will be very important as network-based publishing becomes pervasive. Non-textual approaches to information discovery should also be explored. Furthermore, the content routing architecture needs to be tested in larger environments, realistic information hierarchies should be built, performance could be improved, and there is more to learn about constructing content labels. This section outlines future work in each of these categories.

4.1.1 Legal and Economic Issues

This thesis has concentrated on the technical issues of bringing together people and data. However, this raises many legal and economic questions. Retrieving and copying data becomes trivial in the environments constructed here, raising the problem of intellectual property rights. The delicate balance of authors (and their incentives to produce intellectual property in the first place), publishers, and libraries will need to be drastically restructured. See [54] for a summary of the legal and economic principles at issue. Furthermore, there must be some way for people to make money producing and supplying

the information, so there must be a way to levy and collect charges. If users are being charged, then there must be some authentication and protection mechanisms. See [62, 23, 38] for some recent approaches to these problems.

4.1.2 Visualization

In addition to the tools provided by a content routing system, a user will want other techniques to become familiar with a very large information space. Techniques for helping the user visualize the information space, especially using graphical representations based on spatial and physical metaphors, can be extremely valuable [49]. This involves more sophisticated user interfaces as well as careful analysis of what information will help users the most. For example, a representation of documents as points in a space where distance reflects some similarity metric might be useful, and would require that the system be able to compute a similarity measure over user-selectable subsets of the document space. Given the landscape metaphor of page 12 in Section 1.1.1, it would seem useful to investigate a possible synergism between information visualization and geographic information systems.

4.1.3 Larger Environments

Content routing system tests with many users, much more data, and larger sets of servers are necessary. While 500 servers represents more than a toy collection, it is still quite small compared with the current and future resources of the Internet. Experience with larger populations of users will ultimately determine the value of the architecture. Such large scale tests will require more useful data, and this will require supporting other sorts of end-point information providers. The ability to assemble HTML documents into collections would be very valuable, and tools like the World Wide Web Worm may be helpful in building experimental platforms. Using automated techniques for building hierarchies, *e.g.*, the ideas of Scatter/Gather, would also be very helpful (see below).

Information providers ought to construct their own content labels. This will facilitate

the exploitation of larger sets of data sources. It would be valuable to have information providers run automatic content label extractors, this would allow consistent content labels to be built and would minimize the work required of managers of information providers. However, it is valuable to be able to update the algorithms used for content label construction. Rather than require the installation of new code periodically, information providers could, for example, be asked to run a program every day that downloads the actual content label generating code via the World Wide Web. The code would be available via a URL that the content router could change. This URL could even map to a program that distributes different content label generating code depending on the site making the request. Of course, some information source managers may want to affect the content labels produced for their systems. Balancing these interests without placing undue burdens on any node (or its manager) will present interesting engineering trade-offs.

4.1.4 Building Hierarchies

Organizational Techniques

The work of this thesis has used only *ad hoc* hierarchies of information servers. Other arrangements should clearly be tested. Realistic hierarchies without overly restricted branching factors will require additional data sources. Hierarchies constructed on the basis of principled subject classifications, market principles, and data-driven clustering (Scatter/Gather) are all very important. To build a data-driven hierarchy, one could take a listing of all the documents in the available WAIS servers (using the catalog files) and available HTML documents (using the Web Crawler or some other tool) and then use the Scatter/Gather software to build a static hierarchy. Then one could experiment with content labels derived from the centroid documents of each cluster produced by Scatter/Gather as well as other techniques. An interesting idea here is that once a global hierarchy is constructed, it may not require frequent changing. Perhaps it could be updated every year or so. New documents are fed in at the top of the hierarchy, and an

analog of query routing (document routing) will deliver the document to its appropriate cluster. The new document will be indexed, and it may alter the cluster's content label which may in turn produce changes that ripple back to the top of the hierarchy. I am currently investigating the construction of such a system.

Topologies

In large networks that are not strictly controlled, there will be items that appear more than once in the network. These items may be independent copies on end-point information servers, they may be the same copy available via different routes in the hierarchy, or they may be different versions of the same item. Figure 4-1 shows a content routing system where some documents may be reached via different paths through the network. (Note that a data-driven hierarchy would presumably place copies of an item in the same cluster.) There are a whole range of issues to be dealt with here. Users may not want to know about multiple copies, and so they should not appear in result sets. On the other hand, some copies may be more authoritative than others, so particular copies may be preferred. It may actually be useful to know about different versions of software or software upgrades. If one is looking up a document referenced in a paper, it may be important to find that particular version, not the latest. Jeremy Hylton [28] is already looking at this problem, which librarians have termed *deduplication.*

The content routing architecture needs to be tested in environments with more complicated meshes, *i.e.*, in hierarchies that are not mere trees. This will raise all the issues of deduplication mentioned above. Multiple copies of data will also affects the system's suggestions for a query refinement operation. Term collocations in the same document ought not be counted more than once.

The content routing system implementations have not had to deal with hierarchies with cycles. As mentioned in Section 2.3.1, it is a simple matter to handle cycles.

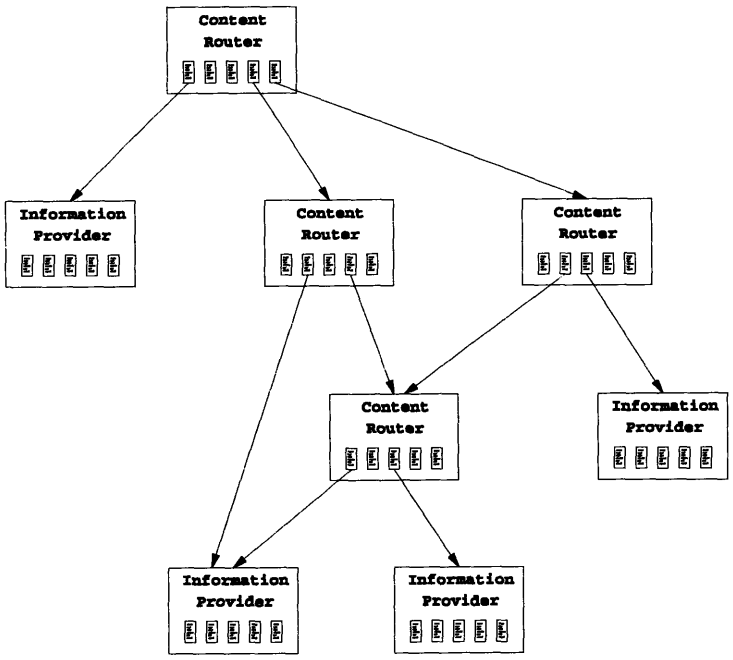


Figure 4-1 A sample hierarchy with non-unique paths to documents.

Scaling Issues

It appears that the query routing feature can scale well. However, it remains to be seen how progressive information loss will affect the utility of the system as hierarchies grow in height. The policy has been to allow each information provider (including content routers) to decide what information to include and omit in content labels. This decision may be affected by the demands of the content routers with which the information provider is registered. An information provider that participates in multiple information hierarchies may have a different content label for each hierarchy.

Will a content routing system be useful as data is lost? This problem is ameliorated to some extent by building bushier hierarchies, but eventually it must be addressed. It would be very interesting to explore a more rigid set of rules for information loss, such as the name server model discussed in Section 2.3.3. This provides a distributed global index semantics with a form of progressive discovery of query terms. My intuition about this is that once the free text attribute is eliminated, there is really only a need for one layer

of content routing. Also, the system may force users to submit queries over attributes for which they do not know the relevant data, *i.e.*, users are generally better at guessing the text terms in documents they want than they are at guessing other attributes (like file owner, author, or location).

As terms are trimmed from content labels as information is propagated up the hierarchy, will the user receive adequate guidance from query refinement? Keeping a mapping from all terms to all documents is the same as maintaining a global index. However, query refinement is really concerned with term co-occurrence, so a more compact representation of this would be very useful. Collapsing several documents into a single virtual document (document clustering) will reduce the size of the mappings used in the current implementations at the expense of creating false term collocations. It would be useful to see how one can trade off data set size against utility. We have experimented with treating all documents at a server as a single document, but this made query refinement nearly useless at reducing result set size (though it does reduce route set size, see Section 2.3.2). Perhaps a system that allows estimation of term collocation similar to the scheme used for routing in GlOSS would be useful. Using somewhat inaccurate estimators for query refinement would not affect query routing, and assuming enough terms are recommended to the user, the suggestions might still prove useful. This is an area of on-going research.

4.1.5 Performance

Replication

Replication is an important idea that should be included in future content routing systems. Replication has three important applications: it provides a mechanism for reducing the load on network hot spots, it reduces operation latencies, and it makes resources more highly available. As network-based information discovery systems become more popular, the demand for certain resources will increase. Replication allows several systems to support this demand. Because systems are widely distributed, a client will see much better

performance if there is a nearby (or more lightly loaded) replica. Finally in any large distributed system, there will always be systems that are unavailable. This is a great annoyance to the user and raises many problems in the design of the interface (*e.g.*, client code may have to save queries and poll previously unavailable servers). Replication can help to mask the vagueries of host downtime and network problems from the user and the client code.

Caching

If users run queries over and over, or if new queries can incorporate the results of past queries (even if submitted by different users), it may be useful to provide some form of caching. Clients may cache results, but also nodes all along the network may find it useful to cache certain query results. To support this, servers and clients might keep a history of queries and use this information to choose which data to cache. This naturally raises all the issues of cache replacement strategies and invalidation.

New Algorithms and Data Structures

The prototypes presented in this thesis made use of very simple algorithms and data structures that were chosen based on the ease of implementation given the set of tools at hand. It is very clear that performance both in space and time could be improved immensely by better algorithms and specially designed data structures. For example, query refinement is very slow in the prototype systems because it does many string operations that result from the simplistic use of WAIS indices. Hashing or interning query terms and using integer document identifiers would allow for a more compact and faster database. These techniques would be simple to implement given that WAIS catalog files provide document identifiers, which could have tags added to indicate the server that has the documents. I am currently working in this area as well as in the area of a compact representation of query refinement information for content labels.

It may also be possible to use different mechanisms for smaller content routers. A

smaller content router may not require the full generality of the indexing technology chosen for our prototypes. An example of a technology for smaller information systems such as personal workstation files with considerably less space overhead, though at some penalty in response time, is explored in [32].

Yet another direction for future work involves expanding query semantics to include inexact matches. This would help to find documents in the presence of spelling errors. Spelling errors in queries is important, but can be helped with local spelling correction software. The harder problem involves occurrences of misspellings in documents. Many misspellings are visible in the WAIS documents available from our prototype. See for example the query refinement suggestions that contain misspelled terms. This quality control problem is becoming more acute as informal and personal publication via the Internet becomes more common. Inexact matches have been handled to some extent in [32, 7, 8].

4.1.6 Content Labels

More research is needed to determine what data should appear in content labels. This will require more data from the information providers, as described above. It is also important to determine how to propagate terms up a hierarchy. My intuition is that the use of high frequency terms will be useful, but it is worthwhile exploring the use of centroid documents as well. It is important to decide how to treat synthetic attributes in content labels, *i.e.*, whether they are apparent in the content label or simply added in to the refinement data as if they occur in every document. Connection information (like `hostname`, etc.) may or may not be separated from other synthetic attributes. Finally, a compact representation of query refinement data is essential.

4.2 Summary

This thesis has presented a new, hierarchical architecture for information discovery and retrieval in large networks of information servers. The content routing system design in this thesis:

- combines offline data organization with interleaved use of browsing and searching at various levels of granularity.
- supports distributed processing.
- exploits heterogeneity of data sources.
- addresses the metadata issue by showing how to automatically generate content labels that describe the information exported by a server.
- provides key abstractions (collection documents) and operations (query refinement, incremental and global search) for effective information discovery and retrieval in large, distributed networks.

Four prototype implementations have explored and verified the design of this thesis. These prototype implementations have answered pragmatic questions about content routing systems and produced the following conclusions:

- A system can help a user cope with vast amounts of information by providing some organization of the data and automatically suggesting improvements to users' queries.
- It is possible to route queries to a meaningful subset of a large number of information servers.
- It is possible to define metadata structures, content labels, to organize the information space and provide other assistance to users.

- The content routing architecture can successfully interoperate with pre-existing systems.
- It can be practical to extract useful content labels from hundreds of disparate, incompatible servers.
- A hierarchical content routing system appears to scale well.

The potential branching factor of a content routing hierarchy is higher than I had thought in the design phase: it now appears that the present implementation technology can easily support thousands of remote information providers (an order of magnitude more than I expected). I now understand better what a content routing system requires of underlying communication protocols, namely, a high tolerance of varying latencies. Experience using the system has shown that browsing and query refinement are essential elements in exploring an information space. The most surprising result is that extremely simple content labels are effective for query routing and refinement. Furthermore, even naive query refinement algorithms are quite useful.

Appendix A

SFS-based Implementations

This appendix provides more details about the design and construction of the NFS/SFS-based content routing system implementations. There were two implementations each built as modifications to the Semantic File System implementation. This chapter describes the very simple user interface, then describes the implementation strategies of the two implementations.

A.1 User Interface

The principle idea of the SFS-based prototypes was that the content routing system would appear as an SFS server (with some semantic extensions) and would behave as an SFS client when communicating with remote information servers. Thus, the user interface was based on path name queries. While path name queries may be syntactically awkward, even unmodified directory editors, such as DIRED in EMACS [65, pp. 92–101], can work quite nicely. A custom graphical browser written in TCL by Ron Weiss was very helpful.

The content router implementations incorporated many simplifications in the design. They supported only conjunctive queries. The first SFS-based content router only provided automatic recursive search. That is, every query was routed to relevant information providers, and the results from the leaves were merged and presented to the user. It was

suggested that users might gain some advantage by being able to browse the content labels.[1] This started my thinking about the model of progressive discovery.

The second SFS-based content router did not provide recursive search, only the *expand* operation of Chapter 2. The idea was to explore the notion of progressive discovery. This more complicated interface is described here.

The user performed an expand by inserting a special attribute into the query (as will be described below). Thus, the user's query was seen as a universe specification (that defined the route set), and a query. The query, of course, could contain further universe specifications and queries. The prototypes did not forward the terms from the universe specifications to the remote servers. This meant that information providers did not have to filter out synthetic attributes that were in their content labels but not in their databases. However, it often meant that the user had to repeat terms. The graphical browser helped make this task a little easier.

The SFS-based content routing systems used the same path name query syntax as the semantic file system [21]), to wit,

```
<CRS-full-path> ::=    /<pn> | <pn>
<pn>            ::=    <attribute> | <field-name>
                       <attribute>/<pn>
<attribute>     ::=    field: | immutable-field: | <field-name>/<value>
<field-name>    ::=    <string>:
<value>         ::=    <string>
```

A content router is mounted like any remote file system. A content router accepts queries in the form of path names and helps users locate items of interest in the network. Queries are made of *attributes* that consist of a *field name* (which must end in a colon) and a value. Some attributes help users identify information providers that interest them, and other attributes help users identify particular items at information providers.

Thus, if a content router were mounted at `/cr`, then the users' interactions with the content router would be via path names that begin `/cr`. At that point, the directory

[1]This suggestion was made by Barbara Liskov while I was writing the thesis proposal.

structure of the remote information provider is visible, and the user may browse these directories just as in any distributed file system.

A user could query the subtree rooted at a particular path by inserting attributes into the paths. The server dynamically creates *virtual directories* that contain the results of queries. Recall that an attribute is a pair of a field name and a value. The SFS syntax reserves the character : as the terminator of field names. Attribute fields and values are separate elements of the path name. For example, the following query would be used to list all documents that are indexed by the attribute `text:clinton`, assuming the content router was mounted at `/cr`:

```
cd /cr
ls text:/clinton
```

Attributes are not required to appear in their entirety. A path that ends in a field name corresponds to a virtual directory that contains a subdirectory for every value of that field. This provides the user a *show values* operation. The SFS-based implementations simply listed all possible values and did not provide any query refinement.

The special directory `field:` is always implicitly available, and it contains a subdirectory for each of the indexed fields. This helps a user explore a new information system that may have locally-defined fields. The `immutable-field:` attribute allowed information providers to assert that the value set for those fields did not change frequently (like the owners of the files on a file server).

All content labels were required to have have a single term with the field `library` whose value was the unique name of the information server. At any point, the user could look in the `library:` virtual directory to see all available information providers. Naturally, this feature is impractical on a large scale, or even on the scale of our WAIS content router.

```
% cd /cr
% ls library:
 nyt
 nyt.desc
 ap
 ap.desc
```

In the above example, a user in a UNIX system has entered the content router by changing to the directory where the server is mounted. The contents of the `library:` virtual directory reveals that this content router indexes only two information servers.

The names ending in `.desc` are files that contain content labels for the respective servers. The content labels are visible so the user can retrieve them and browse for useful attributes or learn about a server's contents.

```
% more library:/nyt.desc
library: nyt
library-type: news articles
hostname: brokaw.lcs.mit.edu
hostaddress: 18.30.0.33
location: MIT
location: Massachusetts Institute of Technology
location: Cambridge Massachusetts
location: USA
location: New England
port: 375
administrator: Mark A. Sheldon
administrator: death@lcs.mit.edu
administrator: 617-253-6264
cost: 0
field: author category date dir exports ext imports keywords name
field: newsgroups organization owner priority subject text title
field: type
immutable-field: category owner priority type
label: New York Times wire service articles for the last 90 days (3
      months).
...
```

The other entries in the `library:` virtual directory are themselves directories. Users may enter these directories to interact directly with the corresponding information providers. That is, these directories allow manual routing of queries when users already know what databases they want to search.

Of course, a user may not decide where to send queries based just on the database name. A user may thus construct a query and let the file system suggest where such a query may be productively routed.

```
% cd label:/times/author:/safire/subject:/language
% ls
 nyt
 nyt.desc
```

In the above example, the system has used its content labels to identify a single relevant database. In this case, the query results in a virtual directory that contains only the content label and routing directory for the **nyt** database. The routing directory contains the items at the **nyt** database that match the query:

```
% cd nyt
% ls
 N203306234  N204275730  N205196132
 N204206066  N205045724  N205265952
% head N203306234
type: NYT (Copyright 1992 The New York Times)
priority: Weekend Advance
date: 03-30-92 1719EST
category: Commentary
subject: BC ON LANGUAGE
title: ON LANGUAGE
author:  WILLIAM SAFIRE
text:

       ME: 'DON'T FINISH MY -- --' YOU: 'SENTENCE.'
```

The same result set could have been obtained with the following query:

```
ls /cr/library:/nyt/author:/safire/subject:/language
```

In the first example, the system determined where to route the query based on the value of the `label:` field and on the knowledge that the NYT database supported the

`text:` field. In the last example, the user has manually determined the database at which the query is to be run.

The universe specification was separated from the rest of the query by the special attribute **`break:here`**. Thus, to route find all articles that mention Bosnia in databases that know about Clinton, a user might try:

```
% cd /cr/text:/clinton/break:/here/text:/bosnia
% ls
```

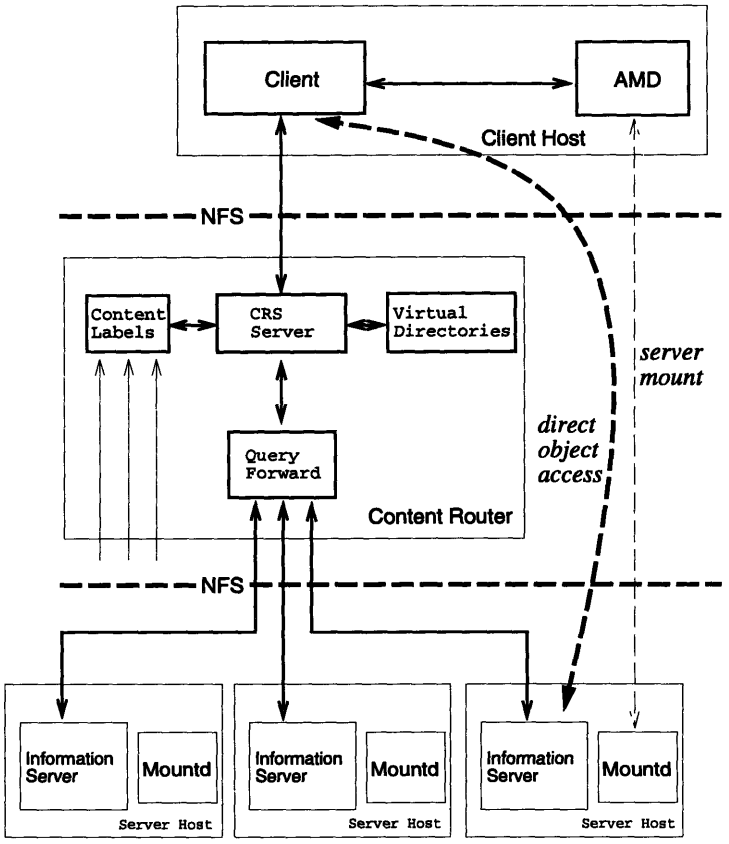


Figure A-1 A prototype content router.

A.2 Implementations

Figure A-1 shows a block diagram of a SFS-based content router. This is a copy of Figure 3-4 reproduced here for convenience.

The content router ran as a daemon that appeared as a (user-level) NFS server on a non-standard port. Clients could mount the server specifying the port number, and then the client kernel would make normal NFS remote procedure calls to the content router. For normal directory and file operations, the content router simply forwarded the requests to the underlying file system.

For virtual directories, query result sets, and remote information servers, the daemon maintained an in memory cache that mapped NFS file handles to nodes in the cache. Virtual directory nodes contained a flag to determine whether they had been filled in.

If an NFS request came in, say a `lookup` or a `readdir` in a non-filled virtual directory, then the system recognized either a *lookup fault* or a *readdir fault.*

A lookup fault for a field or value virtual directory would simply cause the necessary virtual directory node to be built, installed in the cache with an assigned NFS handle, and the handle to be returned. A lookup fault for a file within a value virtual directory (which would be the result of a query) would in turn cause a readdir fault on the virtual directory and then retry the lookup. A lookup in a `field:` or `immutable-field:` directory also caused the directory to be filled.

A readdir fault for an unfilled directory required the following case analysis:

1. The current node is a stand-in for an object on a remote system. Attempt a remote readdir.
2. The current directory is an unrouted query.
   (a) The current directory is a request for the fields we know about. Just fill it in. (Should be a special case of (2) (b) below.)
   (b) The current directory is for an enumerated value set. Call the index system's enumeration routine.
   (c) The current directory is a complete query.
      i. Compute the set of information providers to which the query might plausibly be routed.
      ii. For each such information provider, make a link to its content label and make a link to the query on that information provider.
3. The current directory is an automatically routed query.
   (a) As in 2 (c) (i) above.
   (b) For each information provider, run the query remotely and make links for the results.

The first SFS-based implementation provided only mediated service, that is, the content router mediated all traffic between the client and the remote information providers. Since the remote information providers were also SFS-compatible, these servers could

be mounted and appropriate remote procedure calls could be made to them. The registered servers were given integer identifiers representing their position in a registration table. The indexer was given a copy of the content label whose name was this integer so that queries applied to the local routing database would return this identifier. This implementation read the known servers from the table when the daemon was started, mounted them all, then built an internal array of representations of the NFS file systems and made normal procedure calls (which the file system abstraction would forward to the remote systems). Relevant servers were identified by the integer identifier received from the database system, this was used as an index into the table, and remote operations could easily be carried out.

The second SFS-based implementation was considerably more flexible, and it was to this implementation that the SFS-WAIS gateways were made available. In this implementation, content labels were indexed differently. The transducer that produced the index table from content labels would output a string that could be parsed into the file name, the server's host name, and the service port. The implementation would then parse this string when it received query results from the local database.

The server could be configured either to mediate all traffic between the client and remote systems, or it could allow for the client to contact the remote system directly, bypassing the content router. The contact information was stored in the cached node representing the server and, as above, appropriate RPCs were made to the remote server.

The more typical, and better performing, mode of operation was non-mediated (or direct) processing. Directories for direct connections to remote servers and final query results were made into links that were to be interpreted by the client to allow a direct connection. Clients were assumed to have a directory called `/crs_mounts` which was served by a modified and specially configured version of the amd automount daemon [43, 44, 45]. The links were of the form `/crs_mounts/`*hostname/port/...remote-path*. How this was handled by the automount daemon is described below.

Modifications and Configuration of the Automount Daemon

There was only one, simple modification to the Berkeley automount daemon, and that was to support the ability to mount NFS file systems that used non-standard ports. Apparently, this code had been put into the automount daemon some time before, and then was taken out because it was incompatible with a newer feature (called keepalive). Since then, the code had evolved so that there was a bit of programming to make the general port code work.[2] Changes involved approximately a dozen lines of code.

Using the modified automount daemon did not require that clients know about what servers they would need to contact. Two general-purpose mount maps would suffice to configure the automount daemon appropriately. The first mount map would create the directory for the host name (the first component of the path name after the directory that invoked amd), then it would set up the second mount map to be invoked on subdirectories of the host name directory. The second mount map would collect the next path name component, which was the port number, and then do the mount on the port. Figure A-2 shows the first mount map that extracts the host name from the key and installs the second mount map (`amd.crs.port`). Figure A-3 shows the second mount map which performs the actual mount. Note the long timeout. The mount specifies `ping=-1` because of the above mentioned conflict with the keepalive code. The remote file system is specified as `/` for reasons that are explained below.

One peculiarity of mount map programming is that it is very difficult to build a three level system because of limited argument passing and parsing techniques. This created something of a dilemma. To get the correct root handle of a remote semantic file system, one must mount the correct remote file system. However, there was no way to get host name, port number, and file system through the clumsy mount map code. I made an interim choice rather than try to extend the mount map code. The remote file system that was mounted was always `/`. This meant of course that servers had to export `/`, which

[2]Special thanks to Jan-Simon Pendry for responding to several of my information requests and pointing at the right places to do the modifications.

```
# You may think that we could go ahead and assign rhost:= the current
# key, and then build the map args at each stage of the path, but no.
* type:=auto;fs:=/usr/local/etc/amd.crs.port;pref:=${key}/
```

Figure A-2 `amd.crs`

```
/defaults type:=nfs;rfs:=/
* rhost:=${key/};fs:=${autodir}/${key};opts:=port=${/key},soft,intr,timeo=200,ping=-1
```

Figure A-3 `amd.crs.port`

in a real system seemed unlikely to happen. However, I still needed a way to get to the correct root of the indexed remote file system. The hack was to take advantage of two features of a semantic file system: `field:` is always an available directory, and in any virtual directory ... returns to the file system's root. Thus, links to remote systems had the amd-controlled `crs_mounts` directory, then the host name, then the port number, then `field:/...` then the path or query or whatever was being referenced.

With this infrastructure in place, the automount daemon could be invoked as in the shell script in Figure A-4.

```
#! /bin/sh
#amd -D trace,test ... for debugging.       +---  Also can change the log.
#                                           |
#
/usr/local/etc/amd -a /tmp_crs_mnt -d LCS.MIT.EDU -l syslog \
  /crs_mounts /usr/local/etc/amd.crs
```

Figure A-4 The command to get amd going properly

Bibliography

[1] B. Alberti, F. Anklesaria, P. Linkner, M. McCahill, and D. Torrey. The Internet Gopher protocol: A distributed document search and retrieval protocol. University of Minesota Microcomputer and Workstation Networks Center, Spring 1991. Revised Spring 1992.

[2] D. Barbara. Extending the scope of database services. Technical Report MITL-TR-44-93, Matsushita Information Technology Laboratory, Princeton, NJ, Jan. 1992.

[3] D. Barbara and C. Clifton. Information Brokers: Sharing knowledge in a heterogeneous distributed system. Technical Report MITL-TR-31-92, Matsushita Information Technology Laboratory, Princeton, NJ, Oct. 1992.

[4] T. Berners-Lee. Hypertext transfer protocol. Internet Draft, Nov. 1993.

[5] T. Berners-Lee, R. Cailliau, J.-F. Groff, and B. Pollermann. World-Wide Web: The information universe. *Electronic Networking*, 2(1):52–58, 1992.

[6] T. Berners-Lee and D. Connolly. Hypertext markup language. Internet Draft, July 1993.

[7] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. Harvest: A scalable, customizable discovery and access system. Technical Report CU-CS-732-94, University of Colorado Department of Computer Science, Boulder, Colorado, July 1994.

[8] C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. The harvest information discovery and access system. In *Proceedings of the Second International World Wide Web Conference*, pages 763–771, Chicago, Illinois, Oct. 1994.

[9] CCITT. The Directory - Overview of Concepts, Models and Services. Recommendation X.500, 1988.

[10] CCITT. The Directory - Overview of Concepts, Models and Services. Recommendation X.500, 1988.

[11] D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. In *15th Annual International SIGIR*, pages 318–329, Denmark, June 1992.

[12] P. B. Danzig, S.-H. Li, and K. Obraczka. Distributed indexing of autonomous internet services. *Computing Systems*, 5(4):433–459, 1992.

[13] P. B. Danzig et al. Distributed indexing: A scalable mechanism for distributed information retrieval. Technical Report USC-TR 91-06, University of Southern California, Computer Science Department, 1991.

[14] A. Duda and M. A. Sheldon. Content routing in networks of WAIS servers. In *Proceedings of the 14th International Conference on Distributed Computing Systems*, pages 124–132, Poznan, Poland, June 1994. IEEE.

[15] E. N. Efthimiadis. A user-centered evaluation of ranking algorithms for interactive query expansion. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 146–159, Pittsburgh, PA USA, June 1993.

[16] D. Eichmann. The RBSE spider – balancing effective search against web load. In *Proceedings of the First International Conference on the World Wide Web*, Geneva, Switzerland, May 1994.

[17] A. Emtage and P. Deutsch. Archie – an electronic directory service for the Internet. In *USENIX Association Winter Conference Proceedings*, pages 93–110, San Francisco, Jan. 1992.

[18] E. A. Fox, R. M. Akscyn, R. K. Furuta, and J. J. Leggett. Introduction to special section on digital libraries. *Comm. ACM*, 38(4):23–28, Apr. 1995.

[19] W. B. Frakes and R. Baeza-Yates, editors. *Information Retrieval: Data Structures & Algorithms*. Prentice Hall, Englwood Cliffs, New Jersey, 1992.

[20] D. K. Gifford. Polychannel systems for mass digital communication. *Comm. ACM*, 33(2), Feb. 1990.

[21] D. K. Gifford, P. Jouvelot, M. A. Sheldon, and J. W. O'Toole. Semantic file systems. In *Thirteenth ACM Symposium on Operating Systems Principles*, pages 16–25. ACM, Oct. 1991. Available as *Operating Systems Review* Volume 25, Number 5.

[22] D. K. Gifford, J. M. Lucassen, and S. T. Berlin. An architecture for large scale information systems. In *10th Symposium on Operating System Principles*, pages 161–170. ACM, Dec. 1985.

[23] D. K. Gifford, L. C. Stewart, A. C. Payne, and G. W. Treese. Payment switches for open networks. In *COMPCON '95: Technologies for the Information Superhighway*, San Francisco, California, Mar. 1995. IEEE.

[24] L. Gravano and H. García-Molina. Generalizing GlOSS to vector-space databases and broker hierarchies. Technical Report STAN-CS-TN-95-21, Stanford University Department of Computer Science, 1995. To appear in VLDB95.

[25] L. Gravano, A. Tomasic, and H. García-Molina. The efficacy of GlOSS for the text database discovery problem. Technical Report STAN-CS-TR-93-2, Stanford University Department of Computer Science, Oct. 1993.

[26] H. Hahn and R. Stout. *The Internet Complete Reference.* Osborne McGraw-Hill, Berkeley, California, 1994.

[27] D. Harman. Chapter 14: Ranking algorithms. In W. B. Frakes and R. Baeza-Yates, editors, *Information Retrieval: Data Structures & Algorithms*, pages 363–392. Prentice Hall, Englwood Cliffs, New Jersey, 1992.

[28] J. Hylton. Deduplication and the use of meta-information in the digital library. Master's Thesis Proposal, Department of Electrical Engineering and Computer Science, MIT, Dec. 1994.

[29] B. Kahle and A. Medlar. An information system for corporate users: Wide Area Information Servers. Technical Report TMC-199, Thinking Machines, Inc., Apr. 1991. Version 3.

[30] M. Koster. ALIWEB – archie-like indexing in the web. In *Proceedings of the First International Conference on the World Wide Web*, Geneva, Switzerland, May 1994.

[31] D. M. Levy and C. C. Marshall. Going digital: A look at assumptions underlying digital libraries. *Comm. ACM*, 38(4):77–84, Apr. 1995.

[32] U. Manber and S. Wu. A two-level approach to information retrieval. Technical Report 93-06, Department of Computer Science, University of Arizona, Tucson, Arizona, Mar. 1993.

[33] G. Marchionini and H. Maurer. The roles of digital libraries in teaching and learning. *Comm. ACM*, 38(4):67–75, Apr. 1995.

[34] R. S. Marcus. An experimental comparison of the effectiveness of computers and humans as search intermediaries. *Journal of the American Society for Information Science*, 34(6):381–404, Nov. 1983.

[35] R. S. Marcus. Advanced retrieval assistance for the DGIS gateway. Technical Report LIDS R-1958, MIT Laboratory for Information and Decision Systems, Mar. 1990.

[36] M. L. Mauldin and J. R. R. Leavitt. Web agent related research at the center for machine translation. In *Proceedings of the ACM Special Interest Group on Networked Information Discovery and Retrieval (SIGNIDR-94)*, Aug. 1994.

[37] N. J. Neigus. File transfer protocol for the ARPA network. Bolt Beranek and Newman, Inc. RFC 542 NIC 17759, Aug. 1973.

[38] B. C. Neuman and G. Medvinsky. Requirements for network payment: The NetCheque perspective. In *COMPCON '95: Technologies for the Information Superhighway*, San Francisco, California, Mar. 1995. IEEE.

[39] ANSI Z39.50 Version 2. National Information Standards Organization, Bethesda, Maryland, Jan. 1991. Second Draft.

[40] N. R. C. NRENAISSANCE Committee, Computer Science and Telecommunications Board. *Realizing the Information Future: The Internet and Beyond.* National Academy Press, Wasington, DC, 1994.

[41] Open text to offer free full-text index of web. Press release on Newsbytes Clarinet bulletin board, Mar. 1995.

[42] J. Ordille and B. Miller. Distributed active catalogs and meta-data caching in descriptive name services. In *Proceedings of the 13th Internatinoal Conference on Distributed Computing Systems*, pages 120–129. IEEE, 1993.

[43] J.-S. Pendry. Amd — an automounter. Department of Computing, Imperial College, London, May 1990.

[44] J.-S. Pendry and N. Williams. Amd: The 4.4 BSD automounter reference manual, Dec. 1990. Documentation for software revision 5.3 Alpha.

[45] J.-S. Pendry and N. Williams. Amd: The 4.4 BSD automounter reference manual, Mar. 1991. Documentation for software revision 5.3 Alpha.

[46] L. Peterson. The Profile Naming Service. *ACM Transactions on Computer Systems*, 6(4):341–364, Nov. 1988.

[47] B. Pinkerton. Finding what people want: Experiences with the WebCrawler. In *Proceedings of the First International Conference on the World Wide Web*, Geneva, Switzerland, May 1994.

[48] Y. Qiu and H. P. Frei. Concept based query expansion. In *Proceedings of the 16th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 160–169, Pittsburgh, PA USA, June 1993.

[49] R. Rao, J. O. Pedersen, M. A. Hearst, J. D. Mackinlay, S. K. Card, L. Masinter, P.-K. Halvorsen, and G. G. Robertson. Rich interaction in the digital library. *Comm. ACM*, 38(4):29–39, Apr. 1995.

[50] R. Reiter. On closed world data bases. In H. Gallaire and J. Minker, editors, *Logic and Data Bases*. Plenum Press, New York, 1978. Based on the Proceedings of teh Symposium on Logic and Data Bases, Toulouse, France, November 1977.

[51] G. Salton. *Introduction to Modern Information Retrieval*. McGraw-Hill, New York, 1983.

[52] G. Salton. Another look at automatic text-retrieval systems. *Comm. ACM*, 29(7):648–656, July 1986.

[53] G. Salton, J. Allan, C. Buckley, and A. Singhal. Automatic analysis, theme generation, and summarization of machine-readable texts. *Science*, 264:1421–1426, June 1994.

[54] P. Samuelson. Legally speaking: Copyright and digital libraries. *Comm. ACM*, 38(4):15–21, 110, Apr. 1995.

[55] M. F. Schwartz. The Networked Resource Discovery Project. In *Proceedings of the IFIP XI World Congress*, pages 827–832. IFIP, Aug. 1989.

[56] M. F. Schwartz and J. S. Quarterman. A measurement study of changes in service-level reachability in the global internet. Technical Report CU-CS-649-93, University of Colorado, Boulder, May 1993.

[57] D. A. Segal, D. K. Gifford, J. M. Lucassen, J. B. Henderson, S. T. Berlin, and D. E. Burmaster. Boston community information system user's manual. Technical Report MIT/LCS/TR-373, M.I.T. Laboratory for Computer Science, Sept. 1986.

[58] M. A. Sheldon, A. Duda, R. Weiss, and D. K. Gifford. Discover: A resource discovery system based on content routing. In *Proceedings of The Third International World Wide Web Conference*. Elsevier, North Holland, Apr. 1995. To appear in a special issue of Computer Networks and ISDN Systems.

[59] M. A. Sheldon, A. Duda, R. Weiss, J. W. O'Toole, Jr., and D. K. Gifford. A content routing system for distributed information servers. Technical Report MIT/LCS/TR-578, M.I.T. Laboratory for Computer Science, June 1993.

[60] M. A. Sheldon, A. Duda, R. Weiss, J. W. O'Toole, Jr., and D. K. Gifford. Content routing for distributed information servers. In *Fourth International Conference on Extending Database Technology*, pages 109–122, Cambridge, England, Mar. 1994. Available as Springer-Verlag LNCS Number 779.

[61] P. Simpson and R. Alonso. Querying a network of autonomous databases. Technical Report CS-TR-202-89, Princeton University, Princeton, NJ, Jan. 1989.

[62] M. Sirbu and J. D. Tygar. NetBill: An internet commerce system optimized for network delivered services. In *COMPCON '95: Technologies for the Information Superhighway*, San Francisco, California, Mar. 1995. IEEE.

[63] K. Sollins and L. Masinter. Functional requirements for uniform resource names. Network Working Group, RFC 1737, Dec. 1994.

[64] P. Srinivasan. Chapter 9: Thesaurus construction. In W. B. Frakes and R. Baeza-Yates, editors, *Information Retrieval: Data Structures & Algorithms*, pages 161–218. Prentice Hall, Englwood Cliffs, New Jersey, 1992.

[65] R. Stallman. *GNU Emacs Manual.* Free Software Foundation, Cambridge, MA, Mar. 1987. Sixth Edition, Version 18.

[66] NFS: Network file system protocol specification. Sun Microsystems, Network Working Group, Request for Comments (RFC 1094), Mar. 1989. Version 2.

[67] S. Wartik, E. Fox, L. Heath, and Q.-F. Chen. Chapter 13: Hashing algorithms. In W. B. Frakes and R. Baeza-Yates, editors, *Information Retrieval: Data Structures & Algorithms*, pages 293–362. Prentice Hall, Englwood Cliffs, New Jersey, 1992.

[68] G. Wiederhold. Digital libraries, value, and productivity. *Comm. ACM*, 38(4):85–96, Apr. 1995.

[69] R. Wilensky. UC Berkeley's digital library project. *Comm. ACM*, 38(4):60, Apr. 1995.