Resource Discovery in an Internet Environment

Peter Deutsch,
School of Computer Science,
McGill University, Montreal,
June, 1992.

A thesis submitted to the Faculty of Graduate Studies in partial fulfillment of the requirement of the degree of Master of Science.

Abstract

This thesis examines issues relevant to the design of distributed resource location systems capable of functioning in a multi-user, multi-computer internet environment. Central to this work is the concept of "resource discovery", that is, the act of discovering the existance of classes of resources, locating specific instances of such classes, and accessing these instances.

The problems of resource discovery in a network of hundreds of thousands, or even millions of computers are fundamentally different from those encountered in a local area network environment. In this thesis, an architecture for a resource discovery service is proposed that allows individual users to locate and access arbitrary collections of resources throughout a large-scale internet. Resources served by this system may be typed and treated as individual objects in a distributed, automatically maintained information system.

Cette thèse examine la création des systèmes facilitant la localisation des ressources sur les réseaux de grande envergure. Le problème adressé, la localisation des ressources, est composé de trois problèmes individuels: trouver des serveurs des ressources, trouver des ressources individuelles, ainsi que procéder à l'accès aux ressources trouvées.

Les problèmes de localisation des ressources sur les grands réseaux (qui consistent des milles, ou meme des millions d'ordinateurs) ne sont pas les mêmes que ceux rencontrés sur les réseaux locaux (les "LAN"). Cette thèse présente l'architecture d'un service des réseaux qui permet aux utilisateurs de trouver et manipuler des ressources sur les réseaux des réseaux (les "internets"). Dans cette architecture, les ressources sont traitées comme des objects qui sont automatiquement ramassés dans les bases des données distribuées. L'utilisateur peut ansi poser des questions aux bases des données.

Acknowlegements

I would like to thank Alan Emtage for the many valuable conversations we have shared over the past three years. The indexing services portion of the Resource Discovery architecture presented in Chapter 4 is based in part upon ideas that grew out of our collaboration on the archie system and its follow-ons. Alan's implementation of the original archie system provided a valuable testbed for trying out our ideas and continues to provide a valuable service to the Internet.

I would also like to acknowlege the cooperation of Prof. Clifford Neuman of ISI, Prof. Mike Schwartz of the University of Colorado, Prof. Peter Danzig of USC, Tim Berners-Lee of CERN, Mark McCahill of the University of Minnesota, Ed Vielmetti of MSEN Inc., and Brewster Khale of Thinking Machines Corp., all of whom shared conversations, insights and information about their individual projects that permitted me a better understanding of the issues addressed in this thesis.

My gratitude goes to the staff of the School of Computer Science at McGill for their valuable cooperation and support. In particular I'd like to thank Luc Boulianne, Bill Heelan, Wanda Pierce and Christopher Rabson of the School's technical staff for their professionalism and friendship over the years. I'd also like to thank Vicki Kierl and Lorraine Harper of the School's administrative staff, both of whom helped me in many ways to see this through.

Finally, my wife France and my children Jessica and Jérémy deserve much of the credit for helping me to see that this work was completed. Their support sustained me throughout and I own more to them than I can repay.

Table of Contents

Chapter 1
Statement of the Problem

Introduction

This thesis examines issues in the design of distributed resource location and access systems capable of functioning in a multi-user, multi-computer internet environment. Central to this work is the concept of *Resource Discovery*, that is, the act of discovering the existence of classes of resources in an Internet, locating specific instances of such resources, and accessing those resources. Such resources include but are not limited to files, active processes, hosts and peripherals.

As collections of computers scale up from small collections of machines on single networks (so-called local area nets, or LANs) to large-scale collections of networks (wide area networks, or WANs) the problems of resources discovery change. In particular, the broadcast or multicast techniques that suffice to advertise, locate and access resources such as file servers and printers or other peripherals break down in an environment of hundreds of thousands of hosts spread across thousands of networks [Blaze90].

There are a number of reasons for this. On many collections of networks broadcast message packets are not forwarded through router gateways, thus discovery techniques that rely on such broadcast messaging will not work. Also, polling or searching techniques, in which a user must search through each potential resource provider, become infeasible when there are hundreds of thousands or even millions of resource providers.

Given that it is not possible to detect the availability of services and other resources using a traditional broadcast message paradigm in such environments, new techniques are needed to allow users to find and utilize resources.

In this thesis, an architecture for a resource discovery service is proposed that allows individual users to interactively locate and access arbitrary collections of information throughout a large internet. The proposed architecture uses a registry paradigm featuring proactive data-gathering to verify the existence and availability of resources. This model presumes a reliable network transport layer, but is independent of underlying transport protocols and can be used either interactively or through automated software tools.

One component of this new system allows arbitrary users to publish information and other resources through the use of *information brokers*. Such information brokers simplify the problems of resource discovery and access by providing a database of attributes

for individual resources that can be queried by users and also by other components of the system.

The proposed design allows a high degree of information hiding and automated or software assisted discovery of information not explicitly stored within the system. It also permits non-privileged users control over their published information while limiting exposure of the host machine to unauthorized access. This architecture also addresses a number of security considerations; in particular, the user of such a system need not be granted general access to the host machine to access the information being served.

What is the Internet?

The term *internet* can refer to any network of networks. The so-called *DARPA internet*, or more commonly simply *the Internet*, began as a research project for the U.S. Defense Advanced Research Projects Agency (DARPA) in the early '70s [Tanenbaum88]. It is now a collection of networks spanning the globe, with a large number of research, educational and commercial networks connected together into a single global internet. The Internet now serves as both a live testbed for on-going networking research and a daily communications tool for thousands of users in fields far removed from networking and computer science.

Estimates of the number of machines connected to the Internet vary, depending upon the definition of connectivity used. Some machines are capable only of exchanging electronic mail messages, and yet may be described as *Internet connected*. Some hosts gateway onto the Internet from networks using differing protocol suites, so offer only a minimal subset of shared functionality. The majority of networks of computers connect to the Internet using the TCP/IP protocol suite [Comer91] and machines thus connected have available the full range of Internet protocols and services. Such machines are often described as *fully connected Internet hosts*.

One easily computed measure of connectivity is the count of host address records within the Domain Name System [Mockapetris87] [Mockapetris87a]. Using this metric, the current Internet is now estimated to consist of at least 730,000 machines coupled to several thousand distinct networks, with current growth running at about 30 percent every three months [Lottor92].

The Internet as Service Provider

In the early years of development, the Internet was used primarily for remote login, electronic mail and remote file transfer. Such protocols have been available for some time. Remote login usually uses the telnet protocol described in [Postel&Reynolds 83], elec-

tronic mail is usually transported using the SMTP protocol described in [Postel82]. File transfer from remote archives is usually accomplished using the FTP protocol described in [Postel&Reynolds85].

By necessity much of the early design effort for the Internet concentrated on such low level issues as development of needed communications protocols and hardware, with little time or energy left for the more abstract problems of providing specific user-level services in the new distributed computing environment that was being developed.

This situation is now changing and a wide range of network services is now available. Services offered to users include distributed file systems, distributed bulletin board services (such as the Usenet news service) [Quarterman86], a variety of on-line library catalogues and an extensive network of file archives [Martin91]. More recent additions include distributed Hypertext systems [Berners-Lee92] and distributed information systems [Strauss89] [McCahill92].

With the growth of the Internet, the need has also arisen for distributed information storage and retrieval systems. Such systems allow users to make available information to others, increasing productivity through "cooperative work tools" [Ackerman90] or offering a way to share resources and coordinate work among multiple users [Kraemer88]

Distributed information systems also find use in system-level support tasks. The Domain Name System (DNS) is used throughout the Internet to translate machine names to addresses [Mockapetris87]. The Simple Network Management Protocol (SNMP) allows network administrators to monitor and operate equipment from across a wide area network [Case89].

Other applications already deployed include multimedia filing systems [Gibbs87] and a variety of distributed computing environments [Champine90], [Peterson90] and [Ousterhout88].

The Resource Discovery Problem

As the number of users and hosts continue to grow both individual users and potential service providers have come to recognize that a major challenge exists in identifying the existence and location of information, service providers and other resources in a distributed environment of this size. This problem, the so-called "Resource Discovery Problem", must be adequately addressed if we are to move towards a true Internet-wide model of resource delivery. In this thesis, a survey of current research that addresses components of this problem is presented. Following this, the design of an information discov-

ery and access system is presented that is intended to address specific components of the Resource Discovery Problem.

Modeling the Problem

Several researchers have attempted to model the problem of locating and accessing information in an Internet environment. Yeong, addressing the problem of networked information retrieval, speaks of "*Discovery, Searching and Delivery*" [Yeong91] while Schwartz defines the problem in terms of "*Class Discovery, Instance Location and Access*" [Schwartz91]. Other researchers provide tools to ease the burden of information management while also providing additional accessing tools [Neuman92] [Kahle89].

Combining these views, and building upon Schwartz' taxonomy of the problem, for this thesis the Resource Discovery Problem is decomposed into the four subproblems of *Class Discovery, Instance Location, Instance Access* and *Information Management.*

Class Discovery

The act of *Class Discovery* refers to the identification of a specific class of resources in a larger community of such resources. Thus a user, in searching for a specific class of service providers, might wish to locate all those service providers offering "anonymous FTP archive sites". Such archive sites are Internet hosts that provide universal access to their collections of information using the File Transfer Protocol (FTP) through the convention of a special user code that requires no user authentication. Such sites currently offer a wide range of information, including technical reports and other publications, software and data and provide one of the few universal methods for sharing information currently available on the Internet.

The result of such a Class Discovery query would be a set of resource providers capable of providing the specified type of resources.

User Views and Resource Provider Views

The above view of Class Discovery is based upon a naming model centred upon host and access method. Such a naming model can be considered an extension of what Neuman has referred to as *Host-Based Naming* [Neuman92a]. Host Based Naming refers to the identification of resources, and in particular files, by naming them relative to the host on which they reside.

Examples of host-based naming systems include IBIS [Tichy & Ruan 84] and to some extent Sun's Network File System [Sandberg et al 85].

The problem with such an approach is that, as Neuman notes:

While relatively simple to implement, host-based naming makes it difficult to organize and to locate information: the first part of a file name (the host) usually has little or no relation to the topic, and as a result, logically related information stored on different hosts ends up scattered across the name space." [Neuman92a, p.19]

One alternative is to provide *Global Naming*, in which all resources are named as part of a single namespace, without specifically naming the host. Such a naming scheme is used in the Andrew File System [Howard88] and Coda [Satyanarayanan90].

The problem with this approach is that resource naming relies upon the name prefix to be unique to identify the appropriate resource server in a distributed environment. Resources that are logically related must presumably also share some component of the naming prefix and this constrains where resources can be stored in an Internet environment.

As a third alternative, it is possible to adopt a naming scheme in which we allow the user to name specific classes of resources based upon how the user wishes to organize the information being sought. Such *User Centred Naming* forms the basis of such distributed systems as Amoeba [Tanenbaum90] and Prospero [Neuman92].

With User-Centred Naming, we allow users to group together classes of resources such as "services related to the genome project", or "information services related to Modern Music". The system would then provide access to the component resources while shielding the user from the specifics of resource location and access method. This User-Centred Naming of services parallels the User-Centred Naming used in the Prospero File System.

Ideally, an information location and delivery service would provide support for a mapping between these multiple approaches to naming. Thus, there would be a means for allowing the user to specify a search on the topic of "Modern Music", which would be translated automatically to a search among an "appropriate" series of information servers indexed by resource providers on the Internet. Alternatively, if the user knew in advance that a particular resource is available via anonymous FTP then queries could be constrained and a search limited only to "appropriate" FTP sites. A mechanism for doing this is described in this thesis.

One technique for Class Discovery is described in [Danzig et al 91]. In this paper, Danzig describes a technique for distributed indexing that provides a mechanism for building autonomous databases that specialize in particular topics and types of queries. In

effect, such a system partitions the potential namespace using a variation of user-based naming to reduce the total namespace to be searched. Such partitioning techniques hold out great potential for reducing the size of the namespace that must be searched when making queries to an entire Internet of resource providers. An alternative approach to partitioning is presented in Chapter 4.

Instance Location

Once the existence of a specific class of resource has been established, a user can proceed to *Instance Location*. The objective here is to resolve a user resource query to specific instances of resources that satisfy the conditions of the query. If the result is to be used for *Instance Access* (see below) then a mechanism must exist to map such instances, if provided as user-centred names, to an appropriate host and access method. The archie system, an indexing service that allows such instance searches to be done rapidly, has been described in [Emtage & Deutsch 92].

The archie system proactively builds a database of information, gathered from multiple sites across an Internet. Users of the system search this database through a variety of access methods by sending queries to a database query engine running on the same host as the database.

The archie system is a resource intensive method for providing Instance Location (the current version does not support a distributed database mechanism, so all information to be searched resides on a single host). Still, it has demonstrated the feasibility of such an approach for searching millions of records in the Internet.

Work has also been done in the pilot archie system to provide database mirroring (in which multiple copies of an identical database are made available at multiple sites). To date, this has proved an effective mechanism for coping with rising demand. The mechanisms to do this are still in their infancy and would benefit from further research. Techniques for maintaining loosely distributed network-wide databases are discussed in [Golding91].

One alternative approach to such automated instance location techniques currently employed by Internet users is to interactively browse through individual information delivery systems, such as Gopher [McCahill 92], WWW [Berners-Lee92] or WAIS [Kahle91]. Such an interactive approach does not scale well.

Other techniques include chaining (i.e. the passing of a query from one search engine to another for resolution) or multicast query generation, in which a single query is sent simultaneously to multiple query engines. Both of these techniques were proposed for

the X.500 Distributed Directory Service. [CCITT88] An overview of the X.500 Directory Service is presented in [Deutsch88].

Given the projected growth of information providers on the Internet, more research will be needed to develop additional techniques for limiting the information space to be searched.

Instance Access

Instance Access refers to the application of an underlying access method to access an instance of a resource. There are a wide variety of access methods available on the Internet (as stated, much of the early Internet research revolved around creating the needed protocols and paradigms for resource access). These include traditional file transfer protocols such as FTP [Postel&Reynolds85], distributed file sharing protocols, such as are used in NFS [Sun89] or the Andrew File System [Satyanarayanan90a] and distributed information search and delivery protocols, such as Z39.50 (used in the WAIS system) [Davis90] or Gopher [Alberti92].

The work presented in this thesis is primarily concerned with addressing the individual subproblems of Class Discovery and Instance Location. The Resource Discovery Architecture that is proposed in Chapter 4 is capable of providing the user with needed host and access method information. The knowledge of how to apply such access methods is assumed to reside in the user clients.

Chapter 2
Previous Work

This chapter summarizes research relevant to the various components of the resource discovery problem. In particular, it examines the design and implementation of a number of specific user services now available on the Internet.

Research in this area in recent years has concentrated on tools offering facilities for instance discovery, instance access and information management. These include:

The Domain Name System (DNS)

The Domain Name System (DNS) [Mockapetris87] was an early example of a network-wide distributed database system. The DNS was designed to translate between virtual *hostnames* and corresponding 32 bit *Internet Protocol (or IP) addresses.*

At its heart, DNS consists of a naming taxonomy that partitions a namespace of virtual hostnames (such as *quiche.cs.mcgill.ca*) and corresponding host IP addresses across a hierarchal collection of *DNS servers*. Each server holds a portion of the partitioned database of hostname/address mappings and is capable of responding to user queries on the subset of attributes and values it contains.

Primarily designed to perform translation from *fully qualified domain names* to *IP addresses*, the DNS could be considered primarily an instance location tool, allowing users with only a virtual hostname to map this to the information needed for access.

In operation, the resolution of a domain name to the corresponding IP address using DNS consists of sending a series of queries to a subset of DNS servers, each one responsible for some portion of the corresponding fully qualified domain name. Although normally this would be a potentially resource-intensive set of operations, such an approach is feasible because in this case each individual query can be sent in a single transmission packet and each query can be satisfied in the server $O(1)$ time using a simple table lookup. Note that not all resource discovery tasks fit this model of short, simple and fast queries.

DNS is also used to distribute information about host hardware, operating system configurations and electronic mail exchanger addresses, and it is possible to query the system for wildcard matches (for example, it is possible to ask for all records of a particular type matching a particular string). Thus, it does provide a basic, if primitive class discovery mechanism, as well.

DNS has been an operational success, having expanded continuously since its inception to now cover over 700,000 machine names. Despite this success, there are some problems with both the basic architecture and the specific implementations now in service on the Internet.

Maintenance of the system is distributed, with the required information entered into flat text files (usually by hand) at the site of each authoritative subdomain server. This can lead to inconsistencies and errors in the database that can only be corrected through human intervention. There is no internal consistency checking of this information by the system itself (for example, to verify that registered hosts actually exist on the net).

Another problem can arise during operation. If the authoritative server for a particular subdomain becomes unreachable then users will find that they cannot perform hostname to address conversion. In this case, users can find themselves unable to access a host, even though that particular host is available.

This problem can be alleviated by the use of suitably chosen replicating servers (or by bypassing DNS and using the IP address of a host directly, where it is known) but the configuration and operation of these replicated servers is not automatic and is again prone to human error.

Despite these drawbacks, DNS illustrates the feasibility of distributed database applications in an Internet environment for appropriate applications.

Distributed File Systems

Distributed file systems such as the Network File System [Sun89], the Andrew File System [Satyanarayanan90a] and Prospero [Neuman92] allow site administrators to distribute file systems across multiple hosts in an Internet environment.

NFS

The Network File System (NFS) was developed by researchers at Sun Microsystems and in now available from a wide variety of commercial vendors. NFS is a distributed file system that uses the UDP protocol [Postel80] and a stateless file server model to make available information across a network.

With NFS it is possible to export and import individual disk partitions, or portions of partitions, across a network or entire Internet. Once a partition is mounted on a local host as part of an existing file system tree, the distributed nature of the file system becomes transparent to the user (in terms of naming resources. Of course, response times for

accesses can be markedly poorer for network access on a heavily loaded or widely distributed system, making it obvious when a user has crossed a naming boundary).

AFS

The Andrew File System (AFS) is a distributed file system that offers location-independent resource sharing in a distributed environment. AFS implements a single-image file system for all users with no location dependencies, in effect providing a single global namespace for all users on all hosts. AFS maintains performance in large internet environments through the use of a caching/callback mechanism that substantially reduces network traffic.

Prospero

The Prospero Virtual File System allows the user to create customized views of an underlying collection of information. Among its features, the Prospero file system (actually one component of the larger Virtual System Model for distributed computing under development by its author) provides the capability of creating customized views of available files through user specified links. Each link consists of a pointer to information along with information concerning the associated access method for that information.

Prospero thus implements *User-Centred Naming* of the file system, with each user offered the possibility of creating their own view onto the underlying collection of resources. Such user views are themselves a form of value-added processing of the file system information over and above the contents of the individual files themselves. Such a customized view be exported and accessed by others, aiding in both the instance location and information management problems.

It should be noted that Prospero, unlike the other systems mentioned here does not offer its own access method. Instead, each file in the system is maintained as a link, which consists of location information and details of the underlying access method. The Prospero implementation uses this underlying access method when accessing a specific file.

Currently, Prospero supports the UNIX file system, Network File System and Andrew File System access methods, as well as anonymous FTP.

Prospero could thus be seen as a means for mapping from a user centred naming system to either a host/access method centred system (such as anonymous FTP) or a global naming system (such as AFS). The actual Instance Access is handled by implementation of the required access method protocols in an appropriate library.

Internet White Pages Services

Internet White Pages directory services [Sollins89] are intended to provide users with on-line access to user login names, email addresses and other contact information, thus offering the Internet equivalent of a white pages phone book.

WHOIS

A minimal version of such a system (the WHOIS service) is currently maintained by a number of sites, including the Internet Network Information Centre (NIC) [Harrenstien82]. This service provides a number of small centralized databases, each of which covers only a small segment of the Internet user population. There is no means for locating active WHOIS servers, and currently no mechanism ties together the set of servers to allow a user to scan more than one server.

The current WHOIS design does not scale well and there have been several research projects which have aimed at providing a more robust model.

X.500

A White Pages Directory Service project based upon the X.500 protocol is described in [Rose92]. The heart of the X.500 service revolves around two components. Directory User Agents (or DUAs) are client programs that are able to access information providers or so-called Directory Service Agents (or DSAs) using the X.500 protocol. DSAs are organized as a set of authoritative servers, connected through a single hierarchal naming authority.

X.500 naming is organized along geographic lines, with national organizations responsible for their portion of the name space joining the root DSA, and authority for subtrees in this namespace controlled by and served from DSAs located throughout the Internet.

There are a number of management concerns that confront potential information providers in any White Pages service. These include privacy and security issues, the effort required to create and manage the needed databases and problems required in converting a large institution to such an Internet-based services model. These problems are discussed in [Hill92].

A particular problem for current distributed White Pages projects is maintaining the required data in an accurate and consistent state. This is addressed in the X.500 architecture by distributing the authority for operation of the Directory across the Internet through a name registration authority mechanism.

Another concern is maintaining access control and security in an Internet environment. This is addressed in X.500 through the provision of mechanisms for offering both access control lists and data encryption techniques.

The X.500 service has been under development for some time, and a number of public domain and commercial implementations have been developed. However, despite a great deal of research and the deployment of a number of pilot DSA servers, to date the system has not been an operational success.

A particular problem has been the inherent complexity and tremendous effort required to start offering an X.500 server, especially the effort required to convert existing information into a format suitable for existing X.500 implementations. With relatively few operational servers, users have had little incentive to join in the pilot project. In turn, this lack of a significant number of operational participants has limited the amount and quality of information available.

This failure to reach "critical mass", despite the large amount of research and operational testing that has occurred, offers a lesson to would-be designers of other potential Internet services. To reach a wide audience such services should be available with relatively little effort (at least for experimental testing by users) and should offer a variety of useful information from the earliest opportunity.

The designers of other, more recent information systems (such as Gopher and WAIS, described below) have addressed this problem by offering working servers that could be accessed using a minimum of software. For example, trial access to both of these services is available using only a telnet session.

The lack of initial information to bootstrap the service was addressed in both these cases by providing a variety of gateways to existing Internet services such as anonymous FTP archives and archie.

Information Indexing Services

WAIS

The Wide Area Information System (WAIS) is an example of a network-based document indexing system that has proved useful for accessing large collections of textual data.

The WAIS system is based upon the WAIS protocol [Davis90] which is itself an extended version of the ANSI Z39.50 protocol [NISO88]. The WAIS system provides the user with the ability to search for combinations of keyword strings by sending suitable

search stings (using the WAIS protocol) to the appropriate WAIS servers. Each WAIS server offers access to one or more collections of documents (or *sources*, as they are referred to in the WAIS environment).

WAIS users can rapidly perform keyword searches on documents that can be tens or hundreds of megabytes in size, using a combination of full-text indexing and relevance feedback [Salton & McGill 83].

WAIS document servers build an index database for each source, allowing rapid ($O(1)$) matching for keywords. All sources that contain any of the specified keywords are returned, ranked according the distance between matching keywords in the document and the frequency of these keywords. To speed searches and lower the chances of misleading hits, some WAIS client programs remove short, common words such as "and", "or", "the" *etc.* prior to sending the query to the server.

Once a user has received an initial collection of matches to a search, relevance feedback is used to generate follow-on queries through the selection of additional search terms from the initial replies. This technique allows the user to select additional sources for searching based upon responses that have already been received. This leads to a successive refinement of searches based upon relevant replies until, hopefully, the desired information is found.

The WAIS system addresses both the instance location and information management portions of the Resource Discovery problem, but a major shortcoming in current client implementations is the almost total lack of support for Class Discovery. With currently over 200 WAIS servers on the Internet, the principal mechanism for selecting sources is to page or scroll through a single linear list of source names, selecting the desired sources on which to perform a search. It is expected that future research on WAIS will address this shortcoming.

The original design and implementation of WAIS was done by researchers at Thinking Machines Corp, who have made available a prototype public domain implementation that runs on a number of systems. Thinking Machines also markets the only commercial implementation of WAIS available in 1992. A WAIS Support Consortium has been formed to provide additional direction and support for WAIS research.

The archie System

The archie system [Emtage & Deutsch92] is a collection of tools that, taken together, provide another electronic indexing service for locating information in the Internet environment. One identifying feature of the archie system is that the indexed informa-

tion is proactively gathered onto a central site from primary sources on the net by automated tools, with this collection updated on a regular basis. Access tools are provided that allow Internet users to query this database using a variety of access methods.

The current version of archie consists of three parts; the Data Gathering Component, the Database Maintenance Component and the Database Access Component. The first of these is responsible for locating and obtaining the data to be collected, copying the information into the archie system for processing and storage. The second component is responsible for parsing information that has been copied to the archie system and inserting it into the appropriate archie database. The third component is used to receive and process user queries on these databases.

Client programs are used to access the archie databases and perform searches using a variety of access methods. The archie system does not provide an access protocol of its own. Rather, it can accept user queries via an interactive telnet session to a telnet server, through queries to a Prospero server, or via electronic mail. We have recently added support for WAIS to allow users to index and search large text databases.

Although an archie server demands a great deal of resources to operate, the model has reduced the problem of class discovery for anonymous FTP to one of finding an appropriate archie server. The administrators of the archie service are now responsible for locating and tracking specific service providers for this class of problem.

The architecture of the archie system assures users that indexing information is reasonably current and accurate at all times. The system, as currently deployed, acts as a tool for Instance Location and has demonstrated the feasibility of indexing large numbers of Internet sites in a proactive manner. The current implementation tracks the contents of anonymous FTP archive sites and stores several million records gathered from several thousand sites.

Concerns about reliability have been addressed by adopting a mirroring strategy that duplicates the entire archie database onto multiple archie servers. Techniques to exchange database entries are now being developed and include a partitioning strategy in which individual archie sites gather information only on those sites that are "topologically close" (that is, to which the appropriate server enjoys a relatively high bandwidth network connection). The various archie servers then exchange their information using a mirroring algorithm that detects and automatically transfers changed files from one system to another.

Further investigation into mirroring strategies is needed. Also needed in additional research into the benefits of partitioning the information to be tracked, to avoid the neces-

sity of storing all information at all sites. This would address concerns about the scalability of this approach to resource discovery.

Originally designed and deployed as an Instance Location service to track the contents of anonymous FTP archive servers, the archie model can also be used to deploy a Yellow Pages service by gathering the description of Internet resource providers and periodically verifying the existence of such services. Used in this way, the proactive monitoring model provides a potential solution to the problem of Class Discovery.

The architecture for an enhanced indexing system that provides both Class Discovery and Instance Location forms the basis for the Resource Discovery Architecture described in Chapter 4.

Distributed Information Servers

The World Wide Web

The World Wide Web project [Berners-Lee et al 92] is a distributed information access service based upon the hypertext model of information representation [Bush45] [Nelson90].

In the hypertext model information is represented as a collection of hypertext documents, which consist of conventional text augmented by *hypertext links*, which are pointer references to other hypertext documents. In the original hypertext systems all documents resided on a single host, but with more recent implementations multiple documents can be distributed across an Internet [Kahn et al 90].

The WWW architecture consists of three principal components, including *hypertext servers*, a variety of Graphics-based *client programs* and a set of *information gateways* to additional resource providers. The WWW system also provides a common naming scheme for referring to all documents in the system, a common network access protocol (HTTP, or HyperText Transport Protocol) and a common set of data formats for representing information.

In WWW the information servers provide access to three kinds of hypertext documents. *Real* hypertext documents exist as files and can be accessed using the WWW system directly. *Index* documents are actually gateway programs for accessing search engines or other information service providers. These return either real hypertext documents or *virtual* documents, which are created on the fly in response to a query to an Index. Real documents consist of any collection of text and hypertext links, while virtual documents

created as the result of queries to index gateways contain a collection of links to documents satisfying the user query.

The merging of search mechanisms with a distributed hypertext model has produced a system that is capable of addressing both Instance Location and Instance Access in a user-accessible system, but users of WWW sometimes have difficulty in locating specific information and keeping track of where they are in the web of hypertext documents over time. This problem, the so called "hypertext navigation problem" is a common one for hypertext systems developed to date.

The Internet Gopher

The Internet Gopher [McCahill92] is a distributed information browsing and access system that combines many of the features of electronic bulletin board services such as Usenet [Quarterman88], campus-wide information services such as PNN [Strauss89] and distributed directory services such as X.500. Developed at the University of Minnesota, the Gopher project arose out of an operational need for a simple campus-wide information system that could provide support across a campus. The system now finds wide-spread experimental use across the entire Internet.

The heart of the Gopher system is a collection of Gopher servers, which offer collections of information objects, organized around a hierarchal directory of menus. Each information object consists of either a pointer to another menu (in effect a pointer to another directory of information), a collection of text, a pointer to a service accessible through the telnet protocol or a search engine to which the user can direct queries.

Implementation of each of these information objects is hidden from the user. References to menu objects can in fact point to other Gopher servers residing across the Internet and the user can browse these collections of menus without being aware of the transitions from server to server.

Gopher clients use the Gopher protocol to request access to specific menu items. This protocol consists of a simple query-response model. Each time the user issues a query, the client connects to the appropriate Gopher server and sends a selector string (a short ASCII-encoded string identified with each Gopher menu item). The server responds with the appropriate information corresponding to that item and closes the connection. The Gopher servers are stateless, preserving no information between queries. This makes it simple to implement robust servers in an Internet environment. The Gopher protocol is described in [Alberti92].

Access to search engines is implemented as gateways to existing Internet indexing services, including WAIS and archie, or as separate programs to search collections of data local to the Gopher server. The Gopher protocol (and many of the Gopher clients) are capable of transporting and displaying graphics images and playing sound files on suitably equipped terminals.

The Gopher protocol has deliberately been kept as simple as possible. As a result, it is quite easy to implement a Gopher server, and thus offer an initial Gopher service. Although first deployed in mid-1991, there were already several hundred operational Gopher servers on the Internet by early 1992, telling contrast to the difficulties experienced in deploying an operational X.500 directory service.

Class Discovery and Internet Yellow Pages Services

Internet Yellow Pages services would provide a directory of available Internet services and service providers, analogous to conventional Yellow Pages services directories. There have been proposals to offer such services using X.500, it is also possible to build up such a service using distributed naming service mechanisms such as Prospero by creating an appropriate user directory containing references to such services and exporting this view to other users. Although a Prospero view would not provide a Class Discovery mechanism by itself, it would allow users to perform basic instance location once a directory is located. Coupled with an index of such directories, a form of Class Discovery could be performed. Another approach for providing a Yellow Pages Directory service is described in Chapter 4.

Despite an obvious need for Class Discovery services on the Internet, no fully functional Yellow Pages services are yet operational. This may reflect more upon financial and political, rather than technical considerations, as providing such a service for the entire Internet requires a substantial commitment of resources and there is currently no mechanism for providing such services on the Internet on a charged basis.

In addition, a number of the attached networks (including the U.S. National Science Foundation NSFnet backbone that connects many of the U.S. mid-level network service providers) currently prohibit commercial traffic on their portion of the Internet through restrictive "Appropriate Use Policies" that limit or forbid such traffic on those portions of the net [SRI92].

In addition to the political considerations, considerable research still needs to be done concerning the design and operation of such services in an environment of millions of machines and potentially millions of service providers. Such research must address

issues in both resource discovery and the design of distributed database systems that such services will presumably deploy.

Chapter 3
DESIGN ISSUES

In this chapter I examine issues that have influenced the design of the Resource Discovery Architecture described in Chapter 4. Of particular concern is the issue of *naming*, which in this thesis is used to refer to the binding of a *virtual name* to a specific resource available on the Internet. As will be shown, each resource on the Internet can be identified by a *universal resource identifier* (or URI), which describes the access method and physical location of the resource. The successful mapping of virtual names to Universal Resource Identifiers is at the heart of the Resource Discovery Problem as it is modelled in Chapter 1.

Also discussed is the issue of identifying the *information contents* of a resource in an internet environment. This problem exists separately from the problem of resource location and is important when considering, for example, the question of identifying duplicate files with similar names in an set of multiple archives, duplicate files with information that is encoded using multiple encoding schemes or when seeking to identify files in which the information contents are derived from a single parent file. A mechanism is proposed (*Unique Resource Serial Numbers* or URSNs) that addresses these problems

This discussion in turn leads us to the concept of *Virtual User*, an extension of Neuman's Virtual System Model that allows us to more easily create logical collections of resources in an Internet environment. *Information Brokers* are the component of the new Resource Discovery Architecture intended to support both URSNs and Virtual Users.

Some Definitions

As used in this thesis, *naming* is considered to be the act of associating a *name* for an object with a *resource identifier*, which is used to identify a single instance of a resource on the Internet. These might include files, processes, hosts and other resources.

A *name* is a logical identifier of a resource assigned by the user and will normally be expressed in a human-readable form. Examples include traditional filenames (such as "/u/peterd/thesis/chapter_1"), hostnames expressed in domain name format (such as "opus.cs.mcgill.ca") or service names (such as "the archie server at McGill").

A *resource identifier* consists of the physical location information needed to actually access an object. This will consist at a minimum of a *context* (which might alterna-

tively be thought of as an available *access method*) and an associated set of attributes to indicate a specific resource accessible through that context.

As an example, a resource identifier might refer to a specific file available through FTP. In this case, the context would be FTP and the associated identifier would be a hostname, a user ID, the required password and the target filename. Together these supply enough information to access the file using the FTP protocol.

With some access methods the hostname may be either implied or derived from other associated information. For example, if the context is the Andrew File System (which using a global naming mechanism) the access method would consist of some means of identifying this as a reference to an AFS file, plus an associated AFS pathname. The appropriate host will be located automatically by the AFS servers.

A *naming system* consists of a mechanism for translating names into resource identifiers. Thus, the Domain Name System can be considered to be a naming mechanism for mapping machine names (such as *opus.cs.mcgill.ca*) into the corresponding resource identifiers (in this example, *132.206.3.3* is the corresponding 32 bit IP address, expressed in conventional "quad byte" format).

Similarly, distributed file systems can be considered naming systems for translating between file names and specific file addresses (such as the corresponding host, partition and inode number). In most cases the file system will provide an associated access method, although in some cases (such as Prospero) the associated access method is stored as an attribute and used separately.

The *binding* of name to resource identifier is considered to be the act of *publishing*. Once a name has been bound to a resource identifier on the Internet, users can reference and access that resource.

Naming

As already noted, there are a vast selection of resources available on the Internet. These include individual resources such as files, but there also exists resource providers that make available information about collections of Internet resources. Such resource providers include file servers, Usenet bulletin board servers, archie indexing servers and other collections of information. If we are to provide tools for searching for and accessing such resources, we need some way of identifying and locating such resources.

Sollins has examined distributed name management [Sollins85] and Neuman uses User Centred Naming as the basis for his Virtual System Model [Neumann92a]. In this

chapter I examine additional proposals that are currently under development for naming and identifying Internet resources.

Virtual Hostnames and the Domain Name Service

As discussed in Chapter 2, one example of this naming problem (the translating of virtual hostnames into host addresses) was successfully addressed with the deployment of the Domain Name System, or DNS.

As the number of machines and users connected to the Internet continues to grow users face a problem in specifying or naming other individual resources. A single global namespace, although conceptually simple, would lead to collisions and difficulties in organizing and locating collections of related resources. To address this problem, DNS imposes a single hierarchal namespace onto the set of Internet-connected hosts and offers a distributed database access service for translating between this naming taxonomy and individual machine IP addresses.

Using DNS we can specify any Internet user account by combining that user's login name with the appropriate DNS hostname using "domain style addressing". In this system, a virtual name for a particular user code would take the form:

peterd@opus.cs.mcgill.ca

Taken together, this combination of "fully qualified domain name" for a host and the user's identifying name on that host fully identifies a specific user on a specific machine in the Internet. This uniqueness is guaranteed through the use of distributed naming authorities to allocate domain names in a single name space.

Using the Domain Name Service it is possible to translate this fully qualified domain name (FQDN) into an appropriate IP address. Combining this address and associated user account name with a suitable access protocol enables any user to access the files and other resources controlled by that account.

Note that this scheme does not guarantee that we have uniquely identified a single virtual user on the Internet. A virtual user may have multiple user accounts on multiple machines. In addition, under many operating systems now in use it is possible to allocate multiple user names to a single user ID. Still, this scheme does offer a method for uniquely specifying individual virtual user accounts in a distributed Internet environment.

Extending the DNS Naming Mechanism

What we need is to extend the currently available Internet naming mechanisms to allow us to specify an arbitrary virtual naming mechanism for resources.

As a first effort, we might seek to extend the Domain Name System described above to allow us to name individual resources (such as files) made available by a user. Each such individual piece of information can be specified as an single resource belonging to a particular user in the name space using a host-based naming scheme. Thus, a particular item of information could be specified as:

resource.peterd@opus.cs.mcgill.ca

Where "*resource*" refers to some resource made available by the user *peterd* on the host *opus.cc.mcgill.ca.*

For example, individual chapters of a paper belonging to this user could be referenced as:

chapter1.paper.peterd@opus.cs.mcgill.ca

chapter2.paper.peterd@opus.cs.mcgill.ca

. . .

This scheme would provide us with a naming scheme for specifying resources such as files available on specific hosts, but it does not specify an access method that would allow us to access the file. In addition, because it uses a host-based naming mechanism it may be difficult to locate resources that are logically related, if they belong to different users or exist on different hosts.

Virtual Names vs. Resource Identifiers

Neuman addresses this problem in his Virtual System Model [Neuman92a], through the use of User-Centred Naming. In this model, virtual names for resources on an Internet are modeled as a distributed network of directories and links. Links consist of attributes associated with that resource, including the associated access method. The Virtual System Model provides a mechanism for translating from names to specific resource identifiers.

Namespaces in Prospero are actually bound to individual objects. In resolving a reference in Prospero, a name and its associated namespace together are used to resolve a reference.

Synonyms

Using our definition of a naming system it is possible to have multiple names, or synonyms, for a single object. It is also possible to have multiple instantiations of a single resource on the Internet. For example, multiple copies of a particular text file may be stored on several hosts, perhaps identified using a variety of names. Note however, that as used here each resource identifier refers to a single instantiation of such a resource available through a single access method.

Note that this does not prevent having multiple resource identifiers referring to the same resource. For example, a single file from a single host may be available through both NFS and FTP. In this case such a file would have multiple resource identifiers, one per access method.

The Resource Discovery Problem as defined in Chapter 1 can be expressed as the problem of identifying the names of resources that satisfy a given user query and translating those names into a suitable context, host name and access method. This is a two part problem. The first part consists of identifying appropriate resource providers for a given search. The second part consists of searching these resource providers for individual resources, thus translating user-supplied virtual names into the appropriate resource identifiers.

Stated in these terms, the problem appears a trivial one. It is the size of the Internet, and thus the huge number of virtual names and resource identifiers to be queried, that make the problem a challenging one.

Identifying Resources and Resource Providers

It is possible to define a universal encoding method that will permit us to specify resource identifiers for any number of different contexts. Such *Universal Resource Identifiers* (or URIs) would each refer to a specific instantiation of a resource on the Internet, regardless of the access method to be used. Such URIs could also be used to refer to service providers offering collections of such resources.

By defining a single encoding method for multiple environments we gain the capability of exchanging such URIs between a number of different information access systems. This would be useful in systems that use a User-Centred Naming approach and do not attempt to define an access method themselves (such as Prospero), or in systems in which the system's links already contain the needed context and associated access details (such as WWW, WAIS or Gopher).

URIs would also be useful in a system designed to provide Class Discovery through searching of collections of information made available across multiple access methods. If the system were to return the result of searches in a single standardized format then client programs would have the option of supporting access to a variety of resource providers without the need to translate references from one encoding scheme to another.

At a minimum, such an encoding would consist of a context identifier to indicate the environment in which the associated access method is relevant, plus a set of associated attributes appropriate to that context. These attributes would depend upon the context and would therefor have to be defined once for each context to be supported. If the URI is to be used to identify the resource (as opposed to merely locating it) a unique author or publisher identifier would also be needed.

Current Research on URIs

Universal Resource Identifiers have been discussed by Kahle [Kahle91] and Berners-Lee [Berners-Lee92a]. Kahle discusses the format of WAIS document identifiers, which consist of a reference to an originating author, a "disposition" (which refers to the location from which the document is available) and an address, which is a handle by which the document is known at that site. Note, that if used as part of a more general system, only these last two elements are needed for access, along with an indication that the context is a WAIS document ID, while the originating author could be used as a key to additional information, if a suitable method for storing and serving such information were to be defined.

Berners-Lee has proposed a form of "Universal Document Identifiers" for the Internet, in describing the format of WWW hypertext pointers. Under his scheme, such pointers would consist of an access method specifier (that is, a context descriptor) and an access-specific set of attributes. Contexts already specified include anonymous FTP, WWW, WAIS, telnet and Prospero. His proposal is extensible, and additional formats are relatively straight forward to create. In this case, the UDI is used only for encoding access information and doesn't not contain enough information to uniquely identify individual resources.

Work is now underway through the Internet Engineering Task Force (IETF) to define a single standard for universal resource references for the Internet. A preliminary meeting was held at the 23rd IETF in March, 1992 and an ad hoc working group was organized to work on defining a single encoding standard for such references.

Any Resource Discovery Architecture developed should be capable of translating virtual names into specific URIs capable of indicating the associated context and the necessary attributes to access that resource in that context. The specific attributes will of course depend upon the associated context, but clearly what is needed is a single encoding method for such attributes for each context, to allow the creators of clients for various information systems to share information from a single Resource Discovery Service.

The Need for Resource Serial Numbers

Another major problem that confronts the designers of a Resource Discovery system intended to provide Instance Location is that of determining when multiple responses generated in reply to a search query actually refer to multiple names for the same resource (i.e. are synonyms for a single resource residing on a single host) and when they actually refer to multiple resources that match the same query.

In other words, when are they referring to a single resource on the Internet specified by multiple URIs and when are they referring to multiple resources specified by multiple URIs?

A variation of this problem is to be able to identify when two different resources on the Internet (for example two files stored in separate archives) actually contain identical information, despite using different representation formats and differing URIs. For example, one document may contain an ASCII encoding of a document, another contains the same document encoded in EBCDIC. Except for minor differences in the encoding alphabet, these two documents share the same information but would appear, at first glance, to be completely different.

We would like to provide a mechanism that would allow users to address all of these problems without actually forcing them to adopt such inefficient methods as copying candidate files to their machine and to compare the contents of each in turn.

This is particularly important during the early phases of Resource Discovery, when users are performing Class Discovery, and Instance Location. During this phase, individual user queries may result in large numbers of potentially misleading responses. We need to provide users with a means for narrowing the focus of searches and decreasing the number of responses to be verified. Preferably, in most cases, actual access of a document should come later in the identification process, if at all.

In the traditional publishing environment this need is addressed through the provision of unique serial numbers identifying each edition of a specific publication. Every

individual copy of an edition is identified by the same identifier (for books, this is the International Standard Book Number, or ISBN).

For example, the book "The Complete Fawlty Towers" by John Cleese and Connie Booth is identified as ISBN 0-413-18390-4. Each and every copy of this book carries the same ISBN number and multiple copies of the same book can be viewed as multiple instantiations of the same collection of information.

Note that the need to uniquely identify the information content of a resource is separate and independent of the need for virtual names or Universal Resource Identifiers, as described above. URIs are concerned solely with the mapping of a virtual name to a physical access method across multiple environments. The issue here is how to determine information about the contents of specific documents.

It is possible to avoid such problems if on satellite systems we store only pointers to original documents, storing the contents of documents only at their point of publication. Unfortunately, this is usually infeasible, for a variety of reasons: we need to mirror files to improve response time and increase availability, we sometimes wish to provide access to the same information through differing access methods and sometimes the originating site is no longer available to offer the information.

To ensure access in such cases we will find ourselves with multiple instantiations of the same resource with differing URIs. In such cases having a resource serial number that can indicate when multiple references to resources actually contain the same information would be useful.

Unique Resource Serial Numbers

One approach to this problem is to use some form of digital signature, related to the information content of the resource. The objective of such a signature scheme would be to provide a "Unique Resource Serial Number" (or URSN) for each resource available from the Internet. Note that such URSNs are intended solely for identifying the *information contents* of a resource, not the virtual name, the representation, or the access method of that resource.

In examining how such a URSN might be constructed, I note that each resource can be modeled as an object that can be seen as having a single publisher on the Internet. A single publisher in turn will control zero or more such resources. If we model each of these resources as a single information object, we can specify that a URSN will be an attribute of a resource that takes a value unique to that publisher and collection of URSNs.

Each time the publisher creates a new information object (either from scratch or by modifying an existing information object) it would be assigned an URSN number still not used by that publisher. Some mechanism would need to be provided to ensure that this uniqueness is preserved.

If the URSN is assigned at the time of resource creation then whenever a resource is copied the URSN should also be copied with it. Alternatively, it must be possible to identify the publisher and compute the value of the URSN solely from the contents of the resource.

Properly administered, URSNs would have the property that they could identify the information provided by resources across encoding schemes, naming schemes and file system representations. Thus, accessing the URSN of a document interactively through Gopher, programmatically through Prospero or through email to an anonymous FTP archive server should always return an indication that the information content is identical, even given multiple instantiations, provided the document has not been altered other than in representation or encoding.

The Virtual User

In the preceding section we modeled resources as objects belonging to a single Internet publisher.

This simple scheme could be implemented in the current Internet environment by specifying the publisher ID as a *username/hostid* pair, with *username* corresponding to a userid on a particular host and *hostid* corresponding to the fully qualified domain name of some Internet-connected host. This publisher ID, coupled with a simple digital signature scheme would yield a basic URSN system that could be deployed immediately on the Internet.

Although this would work as an initial implementation, this simple scheme has a number of limitations. In particular, a user would be required to start a new series of URSNs each time he or she moved to a new host or a new user account.

To avoid this, we would prefer to allow each individual user accessing the Internet (using one or more username/hostid pairs) to function as a single "resource publisher" on the Internet, generating URSNs in the same series from wherever the user is currently working.

One way to accomplish this is extend the Virtual System Model by postulating the existence of a *Virtual System User*, one whose home (and information about the resources

it controls) would follow the user around the net. This would include the associated URSN generator and other associated information.

The Information Broker

Version control and representation information would still be useful, and a mechanism for obtaining such information is still needed.

Although we might attempt to define a method for encoding such information by defining a suitable structure to the format of URSNs, this is not necessary if we postulate the existence of a "user agent" or *Information Broker* process that can manage the Virtual User's resources and control access to their information.

Once it is possible to identify the author of any signed resource, that user's Information Broker can then be used to manage URSNs and track changes in those resources. Such a broker would also have responsibility for the creation and assignment of additional URSNs. As a bonus, such an information broker can also act as a recipient of information requests from the Internet to the corresponding host. We could easily have it do double duty as both information access manager and URSN manager. Thus, for example, the Information Broker could also serve configuration files for each Virtual User (allowing automated establishment of a new user account anywhere that can communicate with the Broker).

One major advantage of providing a Publisher ID with each document and using an active process as an Information Broker (assuming we have a method for locating the author's Broker) is that we can always contact the Broker for needed version and representation information, when it is needed. This would allow the use of simple checksum schemes for the actual signature (which would be used when comparing signatures for multiple URIs to detect duplication) while allowing us access to the additional information available from the Broker when it is needed, at the price of some additional complexity.

This scheme gives us all those elements originally specified as desirable: content identification (using the signature), version control (using the Broker) and representation isomorphism (using a combination of the URI and URSN). All that is required is that we make available a standardized encoding method for Publisher IDs, along with a method for generating and registering a unique serial number for each resource created. As a final step, we need to provide a mechanism for translating the Publisher ID into the appropriate Information Broker address. This last part is a classic White Pages problem and solutions such as X.500 have already been proposed.

Once we have URSNs available, users comparing the results of queries to a variety of resource providers could rapidly identify when multiple instantiations in fact contain the same information without directly comparing contents. In fact, software user agents could do that for them, sorting and even eliminating duplicates if desired.

Implementing URSNs

Schwartz has suggested using a digital signature scheme to implement URSNs. For example, a digital encryption scheme such as MD5 [Rivest92] can be used to generate digital signatures that, for large files, have a very good chance of being unique.

Although this approach does not *guarantee* the uniqueness of each digital signature, it has the advantage that such signatures can be computed solely using the information contents of a resource (making them easy to apply to existing archives of information). They also do not have to be stored, as they can be recreated on demand, provided the generation algorithm is known.

Such signature schemes do not provide version control or authorship information, but the proposed Information Broker mechanism can be used to provide this functionality. Thus signature schemes can be used as either a transitional mechanism, or as a simple method for providing a subset of functionality as part of a larger URSN scheme.

A naming authority would also have to be established to assign and manage unique user ID numbers. Such a service could be performed using X.500, or alternatively unique Internet user IDs could be assigned by the Internet Assigned Numbers Authority (IANA), which exists to issue values for those protocols or services that require unique values.

A Suggested Format for URIs and URSNs

A URSN would consist of a single Virtual User *Publisher ID* (uniquely bound to that user and namespace) and an associated *publisher's serial number*, unique for that particular Virtual User. Together they would be form the Unique Resource Serial Number for that resource. Any additional information available for that resource would be made available by querying the associated Broker for that Publisher ID.

It will probably be useful to allow multiple formats for the Publisher ID. Initially, these might consist of one of the following: a userID/hostname pair, a valid X.500 username, or a unique registration number assigned by a number authority (such as the Internet Assigned Numbers Authority).

A URI would consist of three elements: *the resource's URSN*, as created and issued by the publisher, *a context* to indicate the namespace of the appropriate access

method, and *a set of access attributes* appropriate for that context. It would probably be appropriate to define the URSN as optional, as in many cases users would be concerned solely with access information, although ideally there should always be a way to obtain the complete reference for a document from any system on demand.

Separating out the identification and representation management functionality like this appears to offer several advantages, not least of which is that it would be possible to retrofit a digital signature on top of most existing resource delivery systems (including the anonymous FTP archives serving millions of files to the Internet) without changes to their basic functionality. This would thus allow us to deploy a partial form of URSNs immediately, as the basic signature can be supplied algorithmically. In future, publishers of resources could deploy their own information brokers to handle the other components of the problem, leading to a complete solution to the problem.

Proactive Discovery Techniques

Proactive discovery techniques, in which automated systems perform a task in the background as a users' agent, have been under study for some time. The Programmers' Apprentice project [Rich&Waters90] used this technique to provide a programming environment that adapted its operations to the user's prior operating history.

Schwartz has used a proactive monitoring model to perform resource discovery on the Internet. Techniques used include monitoring network traffic at specific gateways to detect FTP traffic (and thus the existence of previously unknown FTP archive sites), monitoring email traffic to detect usage patterns to locate pools of related users, and traffic analysis to locate popular service providers. A preliminary implementation of a specific user location service ("netfind"), is described in [Schwartz91a].

A form of proactive discovery is also used in the archie system, in that information is gathered prior to user searches being initiated. This model has been shown to be a useful one in those cases where the cost of individual searches is high, compared to the relative cost of proactive information gathering (for example when attempting to search multiple files archives for a single filename).

In contrast, in the DNS model of direct distributed database searching the cost of individual searches is low (each query to a DNS server takes one packet and an $O(1)$ lookup time and the hierarchal naming scheme allows a rapid partitioning of the namespace to be searched). In such circumstances a query across multiple distributed databases becomes feasible.

Choosing the correct model to use will involve balancing the trade-offs between the cost of gathering and searching large collections of data at a single point and the cost of performing a number of queries at locations across the Internet.

As a final cautionary note, although proactive discovery techniques show promise, care must be taken to ensure that the privacy of users is preserved if such techniques are used in production systems.

Chapter 4

The Resource Discovery Architecture

In this chapter we present the design of the *Resource Discovery Architecture.* This system is based upon a single basic information processing engine that is used to provide the various components of a generalized resource discovery and access service for the Internet. As part of this, I also outline the architecture for an *Information Broker* that provides support for Universal Resource Identifiers (URIs), Unique Resource Serial Numbers (URSNs) and the Virtual User Model, as described in Chapter 3.

The Information Model

In this system information is modelled as collections of typed objects, with each object type featuring a specified collection of attributes. In particular, each object has associated with it a corresponding Universal Resource Identifier and corresponding Unique Resource Serial Number, plus such additional attributes as author, creation date, a brief text description of the object, etc. The Resource Discovery Architecture provides information about individual information objects on the Internet, including their location and access method. Once located using this service, such objects are accessed using existing access methods.

There are two types of servers, *Resource Discovery Servers* (RDS) provide information about Internet service providers and collections of resources. *Resource Indexing Servers* (RIS) provide information about individual resources. In practice, the first type provides a basic Yellow Pages service and the second an archie-like instance indexing and location service.

In addition, *Information Brokers* are used to provide the information associated with each Virtual User. These Brokers can be queried to discover the values of attributes associated with each resource object. In particular, information about the relationship between objects is available (for example, whether a particular object was derived from another object).

Although the proposed architecture does not provide any additional access methods, it does provide a means for determining all the information about the object that is available, through dialogue with the associated Information Broker. Information about each object (including its URI) is served from a series of information indexing servers, all based upon the same basic information processing engine.

URSNs are implemented as proposed in Chapter 3. Each resource has associated with it a Virtual User's Publisher ID (represented by an identifier unique across the Internet) and each Virtual User is associated with a specific Information Broker. It is possible for a single Broker to serve information about multiple Virtual Users.

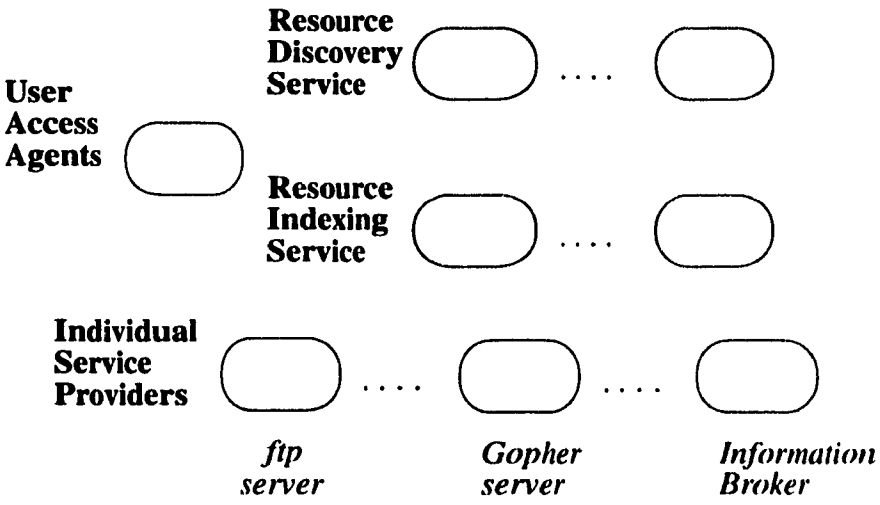


Fig. 1: Resource Discovery Architecture system overview.

The goal for this system was to provide an environment that would allow the user to rapidly search for collections of resource providers that provide needed classes of resources, search these individual resource providers to locate specific instances of needed resources and provide architectural support for locating desired URIs and URSNs.

System Overview

In this section, I present an overview of the Resource Discovery Architecture. This is followed by a description of the basic *information processing engine* that forms the heart of each server within the system.

The architecture of the complete system is shown in Fig. 1 and consists of four parts: The *Resource Discovery Service* (RDS), the *Resource Indexing Service* (RIS), a collection of *Resource Service Providers* (RSP) and individual *User Access Agents* (UAAs), which are user programs that access the other components of the systems.

The Resource Discovery Service

The *Resource Discovery Service* provides the mechanism for **Class Discovery.** Based upon a proactive data gathering model, it allows service providers to register their services with a server. Information needed to access the service (including the Internet host address and the required access method), along with a brief description of the service, are added to a special internal system database, where this information is used to control the data gathering and processing components of the information engine.

Once a service is registered, its registration information and a brief description are also added to a publicly available user database, along with the description information, access information and the current status of the service. The RDS server then periodically verifies the existence and availability of the services tracked, updating the user-readable information as needed.

The system provides a selection of access methods to permit the user to search and browse items available in the user database. Users would be able to query on type of service, service description, or service status.

The RDS acts as a registry of resource providers available on the net, and allows users to query a collection of resource descriptions using a variety of attributes. It is intended to provide proactive location and verification of the existence of specific service providers.

A primary feature of this service is the proactive verification of the information served. Because the RDS periodically connects to each of the services in its registry to verify the service's availability, services that do not respond can initially have their corresponding database entries marked as "Not Responding". If the service remains unavailable for long enough, the entry would be marked for deletion from the database.

Ideally, a description of each service tracked by the RDS would be gathered directly from each service during the verification phase, automatically picking up changes to this information as it is made available. Thus, if a site decides to specialize in a specific type of information, this change would be reflected once the next update is performed.

Using this proactive verification technique, users would be assured access to a reasonably accurate collection of information about resource providers.

Resource Indexing Service

The *Resource Indexing Service* is intended to provide **Instance Location.** Architecturally it is almost identical to the RDS, differentiated more by the type of information

it serves rather than the architecture used for gathering and serving this information. the internal system database again identifies which hosts contain information to be tracked, along with details about how to gather that information and how to process it and store it for access by users.

The primary difference between the RDS and the RIS in operation is that the RDS is concerned with locating, verifying and serving of information about the existence of Internet resource providers (that is, responding to queries about which collections of resources or service providers exist that can be searched to satisfy a given query). In contrast, the RIS is intended to respond to queries concerning specific classes of service provider and responds with information about specific resources.

Both the RDS and RIS are composed of collections of autonomous servers, that are independently maintained and operated. It is expected that in operation both services will consist of a number of individual servers which will specialize in collecting and serving different types of information. This provides both robustness (since it permits the shadowing of popular collections of information) and diversity, since it allows individual service operators to decide which information to gather. It is expected that a variety of different selection criteria will be used at different servers.

It is expected that some RDS servers will concentrate on tracking specific types of servers, while others will track all servers specializing in specific types of information. Details of this specialization will be made available through the RDS itself.

Using a variety of criteria for specifying such services in this way eliminates the need for a single large indexing service for all resources, which would be infeasible as the Internet continues to grow. It also provides a mechanism for offering to users information using both host-based naming and user-centred naming methods.

Note that the basic architecture of an RIS server is identical to the architecture for an RDS server. The primary difference between the two is that RDS servers are intended to track information about service providers, which will in turn provide information about multiple resources, whereas the RIS is intended to provide information about specific resources directly. Where the RDS must verify only that a service provider is alive and functioning to update its database, an RIS server would be expected to copy over specific information for insertion into the appropriate RIS user databases.

Still, in operation the two services are similar. Each server performs discovery to add new sources of resource information into their Internal Hosts Database and periodically connects to resource or service providers to verify the accuracy of the information in the appropriate user database.

Together these two components provide a Class Discovery and Instance Location service, delivering URIs in response to user-generated queries. Once the appropriate URIs have been received by the user, the User Access Agent can then perform Instance Access directly.

Resource Service Providers

In most cases, *Resource Service Providers* are existing Internet service providers, such as anonymous FTP sites, NNTP news servers or information servers such as those offering WAIS, Gopher or WWW services.

In general, individual service providers are tracked in the RDS and part or all of the contents of such services are tracked in the RIS servers. In addition, Information Brokers provide a new type of information service, dedicated to providing information about individual information objects, plus other types of information relevant to individual Virtual Users.

User Access Agents

User Access Agents are client programs used to initiate searches and fetch information from individual users. If the UAA is to be used to both search and retrieve information such clients need to speak both information engine search and retrieval protocols. The search protocol can be any such protocol supported on that information engine (note that the architecture allows the implementation of multiple search and retrieval protocols for accessing individual user databases). The choice of retrieval protocols available in the user's UAA will be what determines whether the resources located using the Resource Discovery Architecture can actually be accessed.

Alternatively, the UAA can be used only to perform searches in the RDA information engine. Once instances of needed resources are located, stand-alone implementations of the appropriate protocols can be used for final access.

Note that the design of the information engine does not require the development or deployment of a specific search or access protocol. The architecture is intended to be extensible, supporting a variety of existing search and access protocols.

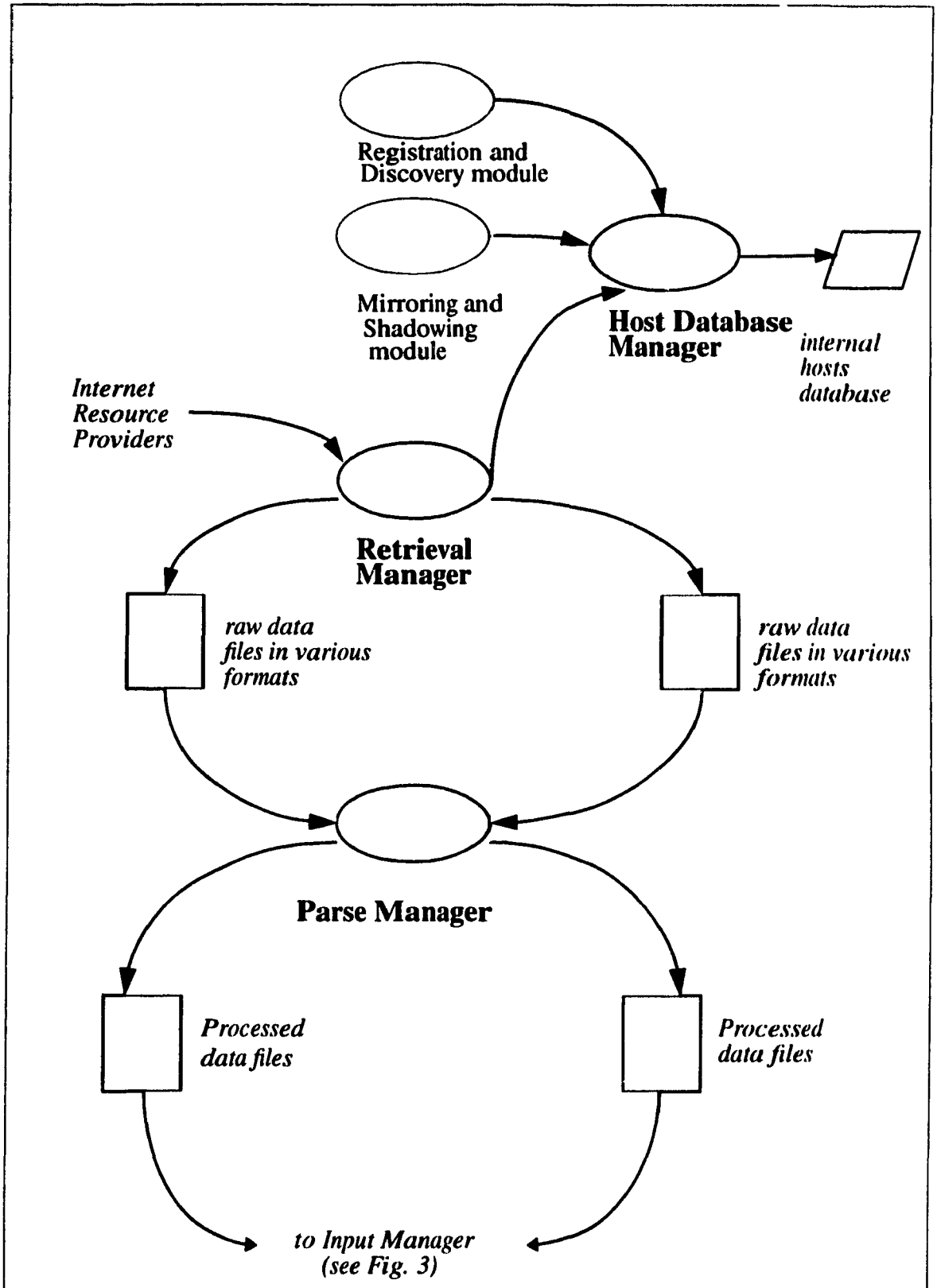
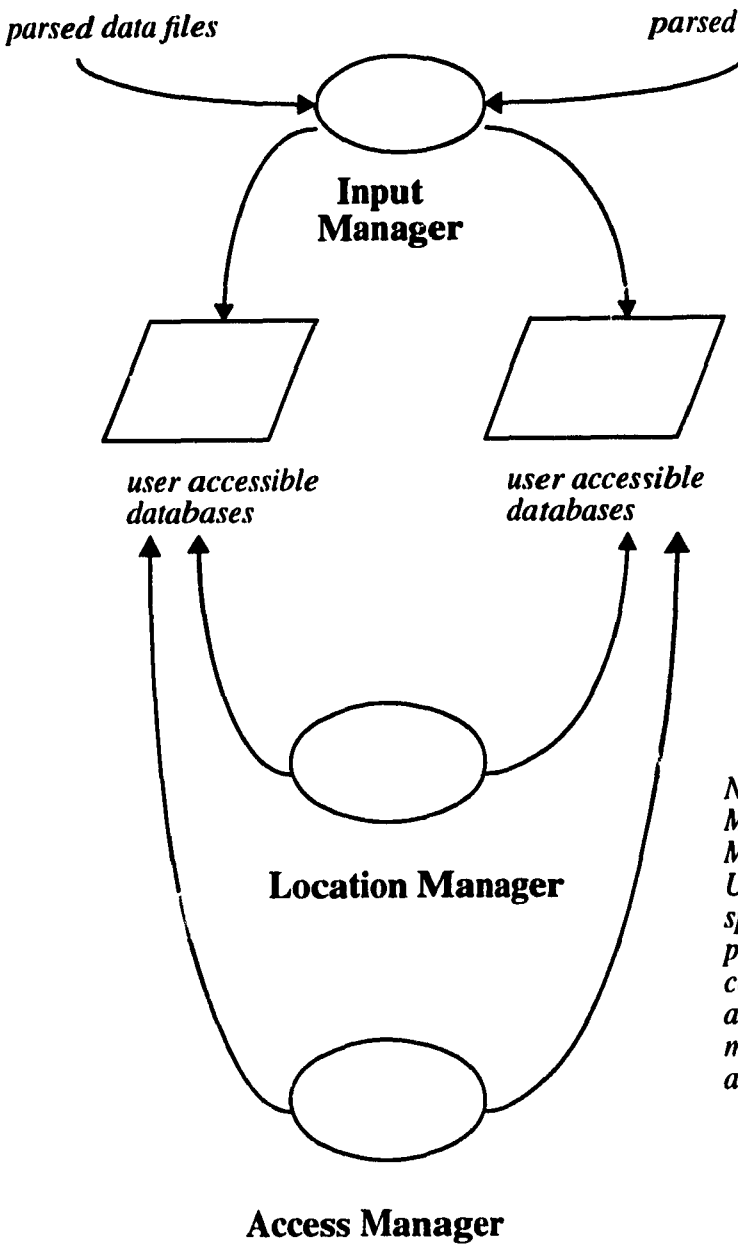


Fig. 2: Information Engine Architecture (Part I).



Note: Both the Location Manager and the Access Manager are called by User Access Agents speaking a variety of protocols, Each component consists of a number of individual modules, each speaking a single protocol.

Fig.3: Information Engine Architecture (Part II).

The Information Processing Engine

The components of the RDA are based upon a single *Information Processing Engine.* The architecture for this engine is shown in Figs. 2-3. The basic architecture of this engine supports servers for both the RDS and RIS services.

The Information Process Engine consists of six components: the *Host Database Manager*, the *Retrieval Manager*, the *Parsing Manager*, the *Input Manager*, the *Location Manager* and the *Access Manager.* These will each be described in turn.

The Host Database Manager

Each information engine maintains a single internal host database. This database contains a list of all service providers known to that engine, along with the information needed to gather and process this information. This includes the access method to use when gathering information (specified as a program to be run by the Retrieval Manager, see below), the time of last access, access frequency, and the type of information to be gathered, including a brief text description for the benefit of the system operator. All access to this database is through the *Host Database Manager.*

A *registration and discovery* module (an adjunct to the Host Database Manager) allows the system's administrator to enter new records into the database. Input to this module may be through an email interface, an interactive program or from a separate resource discovery program. This system could process email registrations automatically but given that such entries are not authenticated, it is suggested that they should be entered but flagged as "not verified". Only after the administrator has checked the entry is a valid request and the service has been verified as existing and operational should information about it be made available to users.

The *mirroring and shadowing* module is used to communicate among Host Database Managers on multiple servers when they wish to use mirroring or shadowing techniques to distribute the database among multiple sites.They communicate information about each site and the information gathered to ensure that all servers have the same user database contents.

The Retrieval Manager

The *Retrieval Manager* is responsible for determining which hosts should be visited to retrieve information for addition to the user databases. It does this by periodically querying the Host Database Manager for hosts that are due for updating, then launching an appropriate data gathering program as a separate process, passing it the name of the target

host and other details as arguments. Any number of such individual data gathering programs can be provided, offering flexibility in the data gathering step.

In the case of the RDS, the data gathering programs are responsible for verifying the availability of the service tracked, and for checking for and fetching service descriptions made available from the service.

In the case of the RIS, the data gathering programs are responsible for determining if the resources being tracked have changed, and if so, retrieving the actual information itself.

The data gathering programs can gather the desired information using any number of techniques. The architecture allows for multiple data gathering programs in a single information engine. These may be anything from simple shell scripts, or simple file copies from a remote host, to dedicated data gathering programs which include data verification and processing routines.

These various retrieval programs are used to create *raw information files*, which contain the information to be added to the user databases in any one of several standard formats. For example, if this service is used to provide an archie-like file listing service, the raw files would be listings in the file formats provided by UNIX, VMS or TOPS-20. If the system was being used to automatically gather information made available via anonymous FTP, it need only copy over the appropriate files.

Once these files are created, the system needs to keep track of the additional processing that remains to be done, passing this information from one component to another. One way this could be done is to prepend a suitably formatted header file to each raw information file created. This header which contains the information needed by the following stages.

The Parsing Manager

Once the Retrieval system has deposited the raw information files onto the system the *Parsing Manager* can select and launch one of a variety of parsing programs to process these raw files into a standardized format for insertion into the appropriate user database.

This partitioning of the data gathering and processing steps allows the one information engine to support a variety of data gathering and access methods. The choice of delivery method for that information is thus left as a choice for the system administrator.

As an example, a dedicated database format would be used to provide the original archie files listing service. This would done because the large amount of redundancy

found in file listings can be exploited to reduce the storage space needed and speed search access over conventional database techniques. Details on how such a dedicated database format can be built are described in [Emtage91].

The Input Manager

The *Input Manager* selects and launches the programs needed to take the parsed output from the previous stage and insert it into the appropriate database. By separating the individual database maintenance programs out in this manner, a variety of database formats can be supported on a single host and any needed special processing can be done on the processed data. If desired, a single collection of information can even be inserted into several different databases.

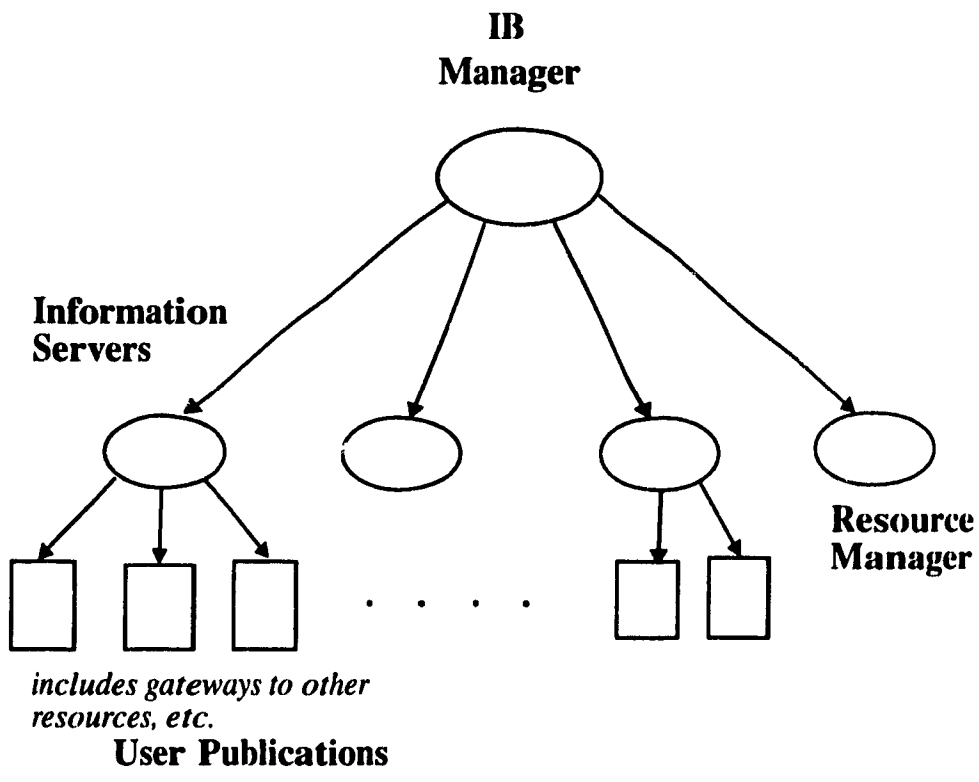


Fig. 4: Architecture of the Information Broker.

The Location Manager

The *Location Manager* is used to provide Internet users with a generalized mechanism for searching within the various user databases. The system provides individual search facilities for each database access method that is supported on that information engine.

Again note that a single information engine may provide a variety of user databases accessed via a selection of access methods. The Location Manager could be seen as providing a gateway service that allows users to specify searches on all of these databases, as well as answering queries about the scope and type of user databases served from that system.

Once a resource or referenced service is located in a particular user database using the Location Manager, the user can access these individual references using an appropriate access method for that user database, through the *Access Manager* (see below). Information needed to access the appropriate component of the Access Manager is returned in the form of a URI. This URI will either point to a resource stored in one of the user databases in the information engine or to a another resource available on the Internet.

The Access Manager

The *Access Manager* is responsible for a collection of programs that together provide access to the individual user databases maintained by the information engine. Each such program would be responsible for implementing one or more access protocols and a single site might provide a variety of such programs providing multiple access methods onto individual databases. A site might offer access methods for such systems as Prospero, WAIS, Gopher, SQL or even anonymous FTP.

Note that it is possible to provide multiple access methods, or even gateways from multiple systems onto a single database. In such cases, the user need not know the underlying storage method used, provided suitable gateways are implemented as part of the various Access Manager processes.

In many cases the results of such a search in the Location Manager will provide all the information needed to allow the user to access the desired resource directly. For example, the results of an archive indexing search sent to the Location Manager of an RIS server tracking anonymous FTP archives would be a series of URIs pointing to individual files on specific hosts. Users do not need to pursue their search further with the RIS, as they now have the information needed to access these files directly.

On the other hand, when searching for textual information served by such a system, the Location Manager may return only the needed indexing information and a pointer to the appropriate Access Manager access program. In this case, the actual fetching of the information would be done using an appropriate protocol to interact with the appropriate server in the Access Manager.

This design provides flexibility to users, as they are not forced into using a single information service to locate and access Internet information.

The Information Broker

Information Brokers are a special class of Individual Service Providers. Information Brokers manage access to publications made available by a collection of Virtual Users and are responsible for assigning and managing URSNs (unless a digital signature scheme is being used, in which case the Information Broker would still act as a central registration service for each Virtual User's published objects, answering queries about all registered resources).

Information Brokers also provide additional access control for the information made available from a specific host. Brokers are publicly accessible from the Internet and can implement specific access control policies for each type of publication served.

The basic architecture of the Information Broker is shown in Fig. 4. It consists of four components, including two types of trusted processes, including *the IB Manager* and a selection of *Information Servers*. All of these processes require privileged access to the host machine to install and modify. The system could also include the actual collection of *User Publications* that will be made available by this site and a separate *Resource Manager* which is responsible for managing URSNs for Virtual Users issued from this system.

The IB Manager

The *IB Manager* is a trusted process that controls all access to the Information Broker system from other hosts. It acts as a "known service" to the Internet for that host. Thus, this process will listen for connection requests on a well-known port and launch the appropriate information server in response to appropriate user requests.

Publication Servers

Publication servers are trusted processes installed by the machine administrators to handle specific types of information for that Broker. Their role is to fetch user-supplied publications, provide gateways to available services and launching additional programs on

the host as needed. They can also perform such tasks as filtering, accounting and access control, as needed.

User Publications

User Publications are created by individual users and registered with the Information Broker's IB Manager process. When users register a publication with the Manager they must specify the location of the publication and the appropriate publication server to be used when serving it to the world. Once registered with the host Manager, these user publications may be considered "published" and available to the network. The Manager can now respond to queries about their attributes of these resources and make them available.

Resource Manager

The *resource manager* is the process responsible for the management of URSNs for Virtual Users from this system. For each Virtual User it tracks it maintains a database of allocated URSNs and associated information. Internet users can query the Resource Manager for information concerning specific URSNs issued.

Additional Architectural Issues

In this final section, I examine a number of remaining issues encountered during the design of the Resource Discovery Architecture.

Resource Provider Discovery

Before a Resource Discovery Service based upon a proactive data gathering model can allow users to search for and discover new collections of resources, some means must be provided for the system itself to discover such resources.

This design employs several techniques. First, we observe that such resource discovery services are themselves "well-known" resources, in the same sense that the root DNS nameservers are well-known. We could thus allow individual users to simply register their resources or collections with the service.

Another technique would be to proactive seek out service providers, using the techniques described in [Schwartz91a]. Schwartz has suggested cooperative monitoring of network traffic to detect the existence of services, monitoring email traffic to detect usage patterns among users, etc. [Schwartz91].

The architecture of the RDS has a separate registration and discovery module, which can be used to implement both of these techniques, if desired. Registration information would come from a variety of sources, including email and interactive input. The Host Database Manager would then verify each entry and add it to the Internal Hosts Database. Notification services based upon electronic mail are already used to maintain electronic mailing lists and provide access to indexing services (such as archie) and anonymous FTP archives.

It would also be possible to use active monitoring and resource discovery techniques, such as those proposed by Schwartz, to detect new services. Such a system could monitor network traffic, Usenet bulletin board postings and electronic mail traffic to detect new services as they become available.

The automated detection and registration of new archives was initially provided by the operators of the archie service, but in certain cases the administrators of such services elected to remain outside of the indexing service. This illustrates that in operation the administrators of such resource discovery services *must* remain conscious of the privacy issues involved. It is probably appropriate to recommend that the operators of services discovered through automated mechanisms, once found, be contacted to verify that entry into the registry is desired before proceeding with registration.

Scaling and Performance Issues

The original archie system was originally conceived as a prototype implementation of an instance location service. Although an operational success (the existing service now handles 50,000 queries per day, at 13 distributed sites) there have been concerns expressed about the scalability of such a centralized indexing services model.

To address these concerns, it is expected that in the future there will be a trend towards more specialized indexing services. By dedicating indexing servers to specific portions of the information space, we avoid potential bottlenecks while also limiting the search for specific types of information, thus improving performance.

An additional benefit of such an approach is that it would permit operators to provide information classified using a user-centred naming scheme. Thus, an indexing service might track "Modern Music" or "Technical Reports", storing URIs pointing to information accessible through a variety of access methods.

When evaluating the performance of a distributed such as the one described in this work, we must distinguish between the performance of the data gathering component and the user agents. From experience with the prototype archie system we can conclude that the principal performance bottleneck for this architecture would be in the database maintenance and access com-

ponents. From initial experiments on the prototype system we conclude that he ability to enter updates and search and retrieve the cached information occupy some 80 percent of the system and leads to the conclusion that the current architecture is primarily bounded by system I/O performance.

One step that was taken in response to this observation was to reimplement the prototype using the mmap() system call, available under SunOS, to allow the use of RAM as very fast storage In effect, by mapping portions of the database into RAM we achieved significant performance improvements with the prototype. Further experimentation in this area is expected to prove worthwhile as we continue to tune database update and access routines.

A number of volunteers have implemented clients for the prototype system, using a library implementing the Prospero protocol supplied as part of the Prospero virtual file system. Using this protocol has allowed us to gained operational experience with the client-server paradigm and the operation of real applications operating over a production Internet. From this experience we concluded that under our implementation of the proposed architecture the primary performance delays were caused by network latency. Transmission errors were not a problem, since the Prospero implementation we used was based upon a reliable datagram implementation but network congestion, especially when traffic was from outside of Canada, dominated query time.

There is a considerable amount of work still to be done further evaluating system performance. The will to a large degree depend upon further in implementation and deployment of practical servers. Work on this continues.

Mirroring Issues

Note that the Resource Discovery Service is not a single monolithic service. Rather, it was intended that the architecture would allow multiple competing service providers to offer multiple views onto the collection of Internet services. This eliminates the need to coordinate a large number of distributed resource providers (which experience with X.500 has shown is difficult, if not impossible). At the same time, it allows a form of "free market" in the partitioning and organization of resource providers to develop, which offers the potential for providing users with a richer and more diverse collection of resources as the number of such Internet resource providers grows.

In this system, each service provider would be able to configure their own RDS and RIS servers to offer their own collections of resources. For example, some providers may wish to organize their information classified in terms of the services provided. Others, might wish to classify using a form of user-centred naming, providing (for example) indexers for such subjects as "Science" or "Music".

Note that both types of partitioning of RIS servers can coexist within a single RDS on the same Internet. Multiple RDS can in turn simultaneously track and provide information about many such collections of RIS servers. The ability to perform rapid interactive searching of large numbers of service descriptions and suitable filtering in the UAA would allow the user to limit the search to the actual indexing services desired.

Given the similarity between the basic architecture of the RDS and the RIS, it is possible to provide a similar degree of multiple partitioning of the RDS service as it grows. Root RDS servers would be able to identify and track collections of services partitioned and identified any number of ways. A relatively modest number of root RDS servers would be needed to provide information about these multiple collections of RDS servers.

This ability to provide multiple partitioning of the collections of information to be searched in both the RDS and RIS components allows the service to scale as the system grows. There are no single points of failure (no single server need remain available, since there may be multiple ways to locate a single service) and the built-in mirroring support makes it relatively easy to set up shadow services for popular servers. Thus the system is expected to be relative robust in operation.

Class Discovery vs. Instance Location

Distributing the indexing service in this manner illustrates the importance of the initial Class Discovery step. A functioning Resource Discovery Service is required to allow the user to locate and access individual Resource Indexing Service operators.

A comparison can be drawn between such indexing services and the role of magazine editors in the traditional publishing industry. A magazine editor acts as a filter, selecting a specific type of information for inclusion in a specific publication. Users are spared the necessity of wading through inappropriate submissions while they are granted access to a timely collection of useful information on the subject of their choice.

The Indexing Services layer is intended to perform exactly this filtering step. Individual indexing services can be established that specialize in various information topics. This reduces the amount of data that must be gathered and restricts the information search space, speeding searches.

Chapter 5
Conclusion

Contributions

This thesis provides a model for the Resource Discovery Problem that permits the design and deployment of a resource discovery architecture capable of addressing the issues of Resource Discovery and Instance Location. In Chapter 1 the problem is decomposed into a set of subproblems that can be addressed incrementally.

In Chapter 2 I include a survey of research activity in the field of Internet information delivery. Included is a description of the majority of relevant projected deployed to date.

In Chapter 3 I address the issue of naming, that is, the mapping of names to specific resource identifiers. I also noted the importance of Resource Identifiers when addressing the Resource Discovery Problem and then presented a proposal for a "Universal Resource Identifier" that could be adopted by existing and future Internet service providers. Doing so would allow the interchange of resource identifiers between disparate systems, easing the burden on creators of information management client software intended to work in an Internet environment.

I also propose the deployment of Unique Resource Serial Numbers addressing the problem of identifying multiple instantiations of identical resources with differing URIs. URSNs can be used to detect such duplicate resources across multiple virtual names, multiple contexts and multiple access methods.

In Chapter 4 I present the architecture for a complete Resource Discovery Architecture that addresses relevant components of the Resource Discovery Problem as modeled in Chapter 1. In particular, the new system address the Class Discovery and Instance Location problems and allows a user to reduce the namespace to be searched when satisfying user queries through the use of class and indexing servers based upon the User-Centred Naming model. This addresses problems associated with the traditional single naming taxonomy used in both DNS and X.500.

My proposed architecture also contains an Information Broker component that provides an additional mechanism for making information available to the Internet with added security and authentication capabilities. This Information Broker will also provide an implementation mechanism for the URSNs proposed in Chapter 3.

Together the components of the system proposed in Chapter 4 provide a viable model for discovering and accessing resources in an large internet environment.

Areas for Future Research

A number of important problems remain to be addressed in the field of resource discovery and access in large internet environments. I will enumerate some of the outstanding issues here.

In particular, more work needs to be done in exploring issues related to the naming of resources. The User-Centre Naming model holds great hope for the design and deployment of practical resource naming and access services, particularly when combined with systems such as the one described in this thesis.

A number of issues in the design and interaction of URIs and URSNs in multiple information systems remain to be explored. In particular, operational issues related to the deployment of a single standard need to be examined. With the potential for an Internet consisting of millions of connected hosts we also need to develop additional mechanisms for narrowing the focus of user queries and thus reducing the namespace to be searched on each query.

Finally, research also remains to be done in the areas of mirroring and providing consistency in distributed databases operating in a large internet environment.

References

Ackerman90 — Ackerman, Mark T.; Malone, "Answer Garden: A Tool for Growing Organizational Memory". Proc. of Conference on Office Information Systems. April, 1990. p 1.

Alberti92 — B Alberti, F. Anklesaria, P. Lindner, M. McCahill, D. Torrey, "The Internet Gopher Protocol, A Distributed Document Search and Retrieval Protocol", University of Minnesota Microcomputer and Workstation Networks Center, Spring, 1992.

Berners-Lee et al 92 — T. Berners-Lee, R. Cailliau, J-F. Groff, B. Pollermann, "World-Wide Web: The Information Universe", Electronic Networking, Meckler, Spring, 1992, p 52.

Blaze90 — M. Blaze, "Issues in Massively Distributed File Systems", Proc. 2nd Princeton University SystemsFest, April, 1990.

Berners-Lee92a — T. Berners-Lee, J-F. Groff, R. Cailliau, "Universal Document Identifiers on the Network", unpublished, but available for anonymous FTP from info.cern.ch as "/pub/www/UDI.ps", February, 1992.

Bush45 — V. Bush, "As We May Think", The Atlantic Monthly, July, 1945, p 101-108.

Champine90 — G. Champine, D. Geer, W. Ruth, "Project Athena as a Distributed Computer System", Computer, September, 1990.

Case89 — J. Case, M. Fedor M. Schoffstal, C. Davin, "A Simple Network Management Protocol", RFC 1098, University of Tennessee at Knoxville, 1989.

CCITT88 — CCITT X.500/ISO 9594-1, "Information processing systems - Open systems interconnection - The Directory - Part1-8", CCITT, 1988.

Comer91 — D. Comer, "Internetworking with TCP/IP, Volume I (Second Edition)", Prentice_Hall, 1991, p171.

Crocker82 — D. Crocker, "Standard for the format of ARPA Internet text messages", RFC822, University of Delware, 1982.

Davis90 — F. Davis, B. Kahle, H. Morris, T. et al 90 Salem, T. Shen, R. Wang, J. Sui, M. Ginbaum, "WAIS interface protocol prototype functional specification", unpublished but available for anonymous FTP from think.com as "/pub/wais/doc/wais-concepts.txt", Thinking Machines, 1990.

Deutsch — D. Deutsch, "An Introduction to the X.500 Series Network Directory Service", June, 1988.

Gibbs87 — S. Gibbs, D. Tsichritzis, A. Fitas, D. Konstantas, Y. Yeorgaroudakis, "Muse: A Multimedia Filing System", IEEE Software, pp 4-15, March, 1987.

Emtage91 — A. Emtage, "The archie System", Masters' Project, School of Computer Science, McGill University, 1991.

Emtage & Deutsch92 — A. Emtage, P. Deutsch, "archie - An Electronic Directory Service for the Internet", Proc. Usenix Tech Conf. pp. 93-110, January, 1992.

Golding91 — R. Golding, "Distributed epidemic algorithms for replicated tuple spaces", Technical Report HPL-CSP-91-15, Concurrent Systems Project, Hewlett-Packard, June, 1991.

Harrenstien82 — K. Harrenstien, V. White, "NICNAME/WHOIS" RFC-812, SRI International, March, 1982.

Hill92 — J. Hill, "The X.500 Directory Service: A Discussion of the Concerns Raised by the Existance of a Global Directory", Electronic Networking, Meckler, Spring, 1992, p 24..

Howard88 — J. Howard, M. Kazar, S. Menees, D. Nichols, M. Satyanarayanan, R. Sidebotham and M. West, "Scale and Performance in a distributed file system", ACM Transactions on Computer Systems, February 1988, p. 21.

Jacob90 V.S. Jacob, H. Pirkul. "A Framework for Networked Knowledge-Based Systems". IEEE Trans. Systems, Man & Cybernetics. 20 No.1, Jan/Feb. 1990, 119-127.

Kahle89 B. Kahle, "WAIS - The Wide Area Information Server Concepts", Technical Report TMC-202, Thinking Machines Inc. November, 1989.

Kahle91 B. Kahle, "Document Identifiers, or International Standard Book Numbers for the Electronic Age", Thinking Machines Corp., 1991.

Kahn et al 90 Kahn, Pau, Meyrowitz, "Guide, Hypercard and Intermedia: A comparison Hypertext/Hypermedia systems", IRIS Technical Report 88-7, Brown University, 1988.

Kill89 S. Kille, "Mapping between X.400(1988)/ISO 10021 and RFC 822", RFC1138, University College London, 1989.

Kraemer88 Kraemer, K. and King, J., "Aids for Cooperative Work and Group Decision Making", ACM Computing Surveys, Vol. 20, No. 2, June, 1988.

Lottor92 M. Lottor, "Internet Growth (1981-1991)", RFC 1296, SRI International, January, 1992.

Martin91 J. Martin, "There's Gold in them thar Networks!", RFC 1290, Ohio State University, December 1991.

McCahill92 M. McCahill, "The Internet Gopher", Proc. of the 23rd Internet Engineering Task Force, San Diego, Ca., 1992, pp 495.

Mockapetris87 Mockapetris, P. "Domain Names - Concepts and Facilities", RFC 1034, November, 1987.

Mockapetris87 Mockapetris, P. "Domain names - Implementation and Specification", RFC 1035, November, 1987.

Nelson90 T. Nelson, "Literary Machines", Sausolito, Ca., Mindful Press, 1990, p. .

Neuman92 B. Clifford Neuman, "Prospero: A Tool for Organizing Internet resources", Electronic Networking, Meckler, Spring, 1992, p. 30.

Neuman92a — B. Clifford Neuman, "The Virtual System Model: A scalable Approach to Organizing Large Systems". PhD thesis, University of Washington, 1992. Department of Computer Science and Engineering.

Peterson90 — L. Peterson, N. Hutchinson, S. O'Malley, H. Rao, "The x-kernel: A Platform for Accessing Internet Resources", Computer, May, 1990, p. 23.

Ousterhout88 — J. Ousterhout, A. Cherenson, F. Douglis, M. Nelson, B. Welch, "The Sprite Network Operating System", Computer, February, 1988, p. 23.

NISO88 — "Z39.50-1988: Information Retrieval Servcice definition and protocol specification for library applications", National Information Standards Organization, Bethesda, Maryland, USA, 1988.

Postel80 — J. Postel, " User Datagram Protocol", RFC 768, ISI, August, 1980.

Postel82 — J.Postel, "Simple Mail Transfer Protocol", RFC 821, Information Sciences Institute, University of Southern California, August, 1982.

Postel&Reynolds83 — J. Postel, J. Reynolds, "TELNET Protocol Specification", RFC 854, ISI, May, 1983.

Postel&Reynolds85 — J. Postel, J. Reynolds, RFC 959 "File Transfer Protocol", ISI, October, 1985.

Quarterman86 — J. Quarterman, J. Hoskins, "Notable Computer Networks", Communications of the ACM, 29(10):932-7, October, 1986.

Rich&Waters90 — C. Rich, R. Waters, "The Programmer's Apprentice", ISBN 0-201 52425-2, ACM Press, 1990, New York.

Rivest92 — R. Rivest, "The MD5 Message-Digest Algorithm", RFC 1321, MIT Laboratory for Computer Science and RSA Data Security, Inc., April, 1992.

Rose92 — M. Rose, "The Little Black Book - Mail Bonding with OSI Directory Services", ISBN 0-13-683210-5, Prentice-Hall, New Jersey, 1992.

Salton&McGill83 — G. Salton, M. McGill, "Introduction to Modern Information Retrieval", McGraw-Hill, New York, 1983, p 321

Satyanarayanan90 — M. Satyanarayanan, J. Kistler, P. Kumar, M. Okasaki, E. Siegelm D. Steere, "Coda: A Highly Available File System for a Distributed Workstation Environment"", IEEE Computer, April, 1990, p. 447.

Satyanarayanan90a — M. Satyanarayanan, "Scalable, Secure, and Highly Available Distributed File Access", IEEE Computer, May, 1990, p. 9.

Schwartz91 — M. Schwartz, "Resource Discovery in the Global Internet". CU-CS-555-91, University of Colorado at Boulder, November, 1991.

Schwartz91a — M. Schwartz, "Resource Discovery and related research at the University of Colorado". CU-CS-508-91, University of Colorado at Boulder, January, 1991.

Sollins85 — K.Sollins, "Distributed Name Management", Ph.D. thesis MIT/LCS/TR-331, M.I.T., 1985.

Sollins89 — K. Sollins, "A Plan for Internet Directory Services (White Pages)", RFC 1107, June, 1989.

SRI92 — "Internet: Getting Started", edited by April Marine, SRI International, Menlo Park, CA, 1992.

Strauss89 — H. Strauss, "University-Wide General-Interest On-line Information Systems that Work - And that You Can Afford", ACM SIGUCCS XVII, 1989.

Sun89 — "NFS: Network File System Protocol Specification", RFC 1094. Sun Microsystems, March, 1989.

Tanenbaum88 — A. Tanenbaum, "Computer Networks", ISBN 0-13-162959-X, Prentice-Hall, New Jersey, 1988.

Tanenbaum90 A. Tanenbaum, R . van Renesse, H. van Staveren, G. Sharp, S. Mullender, J. Jansen, G. van Rossum, "Experience with the Amoeba distributed operating system", Communications of the ACM, 33(12):47-63, December, 1990.

Yeong91 W. Yeong, "Towards Networked Information Retrieval", Tech Report 91-06-25-01, Performance Systems International, 1991.