AN AGENT SYSTEM FOR QUERY ROUTING SEARCH

By
Nicholas Miles Gibbins

A thesis submitted for the degree of
Doctor of Philosophy

Department of Electronics and Computer Science,
University of Southampton,
United Kingdom.

February 2002

UNIVERSITY OF SOUTHAMPTON

ABSTRACT

FACULTY OF ENGINEERING

ELECTRONICS AND COMPUTER SCIENCE DEPARTMENT

Doctor of Philosophy

An Agent System for Query Routing Search

by Nicholas Miles Gibbins

This thesis examines the issues affecting the design and implementation of scalable agent-based systems which use query routing for resource or service discovery. Query routing is a type of informed distributed search in which agents reason about the capabilities of other agents in order to constrain the scope of a query and the cost of processing it.

The technique of query routing bears many similarities to the use of mediators in multi-agent systems. We discuss the relation between mediator-based systems for service discovery in multi-agent systems and the use of query routing in distributed information systems, and present a novel model of the query routing task which we have used to examine the complexity and scalability of a number of commonly encountered architectures for resource or service discovery.

This theory-based approach is complemented by our practical experiences of building query routing systems using our simple agent framework, Phyle. Finally, we perform an empirical study of the behaviour of different query routing systems in order to validate our model, using our simulator for query routing systems, Paraphyle.

This thesis is dedicated to Isobel Stark,
and to the memory of Catherine 'Kit' Manning (1919–2002).

Contents

List of Figures

List of Tables

List of Algorithms

Acknowledgements

I would like to thank the following people for the help and support they have given me during the course of this work:

- My supervisor, Wendy Hall, for her comments and guidance.
- Nigel Shadbolt, for making it possible for me to balance writing this thesis against a full time job.
- The members of the Intelligence, Agents and Multimedia research group, current and previous officemates in particular, for four years of thought provoking and informed discussion.
- The members of `totl.net` for providing welcome distractions when needed.
- Isobel Stark, for keeping me sane even while I was driving her mad.

The work in this thesis was financially supported by an EPSRC research studentship.

Chapter 1

Introduction

During the last decade, the dramatic growth of the Internet and the World Wide Web have only been surpassed by our growing expectations of their effects on our future lives. Hypertext, for years an academic curiosity, is set to become a medium with penetration on a par with television or the printed word. The environment of our libraries and teaching institutions is set to change as the rising tide of digitally available information begins to augment or supplant the traditional physical holdings of ink-and-paper resources. Our direct involvement in day-to-day commercial transactions will decline as we rely on software agents which we have empowered to make buying decisions on our behalf. Behind much of this hyperbole lies an assumption made by many of the more vociferous Internet advocates, that the growth of the Internet is effectively unbounded. Unfortunately, this assumption is incorrect in one key area, that of searching the World Wide Web.

At present, there are two distinct ways of searching the Web. Individual web servers may have a search facility which covers the pages on that server, but this is only of use if the user has prior knowledge of which server is likely to hold the documents which they need. The alternative to this are the web search engines, centralised indices of the Web which are constructed by a brute force traversal of the Web that index each page found. The largest of these services - Altavista[1], Google[2], Lycos[3] and so on - are household names, and provide a way for users to search the contents of many web servers at once. They are a vital part of the web infrastructure, but there are signs that they are increasingly less able to provide an effective service as the Internet and World Wide Web grow.

[1] `http://www.altavista.com/`
[2] `http://www.google.com/`
[3] `http://www.lycos.com/`

Studies of the major search engines by Lawrence and Giles (1998, 1999) estimated the size of the publicly indexable Web to be in the region of 320 million pages in 1998, rising to 800 million pages in 1999. Of these pages, at most 16% (and frequently less) were indexed by the search engines studied in 1999, and this proportion had fallen since the first survey in 1998. The latency of index records also grew, with a typical wait of several months for new pages to be indexed, and as many as 10% of returned links pointing to non-existent resources. Much of this is due to the logistical difficulties of performing a 'full' web traversal in reasonable time. At the time of writing, estimates of Web size exceed two billion pages (Cyveillance, 2000), yet the largest published indices have only just reached one billion pages (Google, 2000). In short, the existing solutions are not scaling well.

The advent of the next generation *Semantic Web* (Berners-Lee et al., 2001), which posits the existence of intelligent agents that can reason using ontologically marked up content found on the Web, is set to highlight the inadequacies of the current approaches to searching large scale distributed systems by reducing the intrinsic granularity of the Web from webpage-level artifacts to entities more akin to facts in a expert system knowledge base.

The task of searching a distributed system for objects with certain desired characteristics, known as *resource discovery*, is found in many domains other than the Web. Hypermedia link resolution in hypertext systems with first class links, white page directory lookup, agent mediation and bibliographic search must all perform resource discovery in a distributed context.

One solution which addresses the question of scalability of the resource discovery task relies on the distributed nature of the system to spread the index-building load. In the Web domain, for example, this would entail each web server indexing its own documents. Searching for documents in this distributed environment requires that the query be evaluated (logically, at least) on each server. If each server passes a summary of their contents to the other web servers, a user's client can locate and query only those servers with relevant content, a technique which is known as *query routing*. We can view this process as the composition of three subtasks: *database selection*, in which a web server containing possibly relevant material is located; *query evaluation*, in which documents matching the query criteria are identified; and *data access*, in which the identifiers are used to retrieve the matching documents. In some domains, the first two tasks are collectively known as *name resolution*, since they turn a descriptive name into an address for a resource.

For query routing to work, the servers must have sufficient a priori knowledge about the contents of other servers to be able to guide the user's client to its destination. This knowledge may be characterised as a belief about the knowledge of another server. Combined with the distributed, ad-hoc nature of the problem, this suggests that *agent-based computing* (Jennings, 2000), which models computation as the social interactions in a group of autonomous processes, is a suitable technique for studying query routing systems. Indeed, the resource discovery task is closely related to the *service discovery* task or *connection* task (Decker et al., 1996) in which agents in a multi-agent system attempt to locate other agents which can provide them with a service that they need. Service or resource discovery services are an essential component of loosely coupled systems like multi-agent systems, as noted by Gasser (2000). Brazier et al. (2001) note that there have been few studies of multi-agent system scalability to date, even though this should be an important consideration when deploying agent systems, and that the majority of agent scalability problems are not agent problems *per se*, but are related to the services provided for name resolution and resource or service discovery.

In this thesis, we study the suitability of query routing as a technique for database selection as part of the resource discovery task, based on an experimental implementation of an agent-based query routing system. In Chapter 2, we review existing systems for resource or service discovery, some of which make use of query routing, and summarise relevant previous work in our chosen problem domains of hypermedia, white pages directories and bibliographic metadata. Chapter 3 contains a model of the query routing task based on existing information retrieval formalisms, while in Chapter 4 we examine the effects of the underlying topology of a query routing system on its behaviour and scalability. In Chapter 5 we present Phyle, an agent-based system for query routing search, and Paraphyle, a simulator for large Phyle systems, and in Chapter 6 we discuss the design of agent-based systems which use query routing, as well as knowledge representation ontologies for our chosen problem domains. Finally, Chapter 7 contains an empirical study of Phyle's behaviour in different network topologies, while Chapter 8 contains our recommendations for building effective and scalable query routing systems, and outlines a number of potential avenues for future research.

Chapter 2

Literature Review

2.1 Introduction

The resource discovery task appears in several domains, and consequently there have been a number of attempts to construct systems which perform well at it. In this chapter, we concentrate on the contributions from traditional computer science, artificial intelligence (by way of multi-agent systems) and library and information science by studying exemplar systems and other previous work from these areas. In addition, we summarise key technologies from our specified problem domains of bibliographic search, hypertext link resolution and white pages directories.

2.2 Distributed Search Systems

Distributed search is used here as a catch-all which encompasses both 'pure' resource discovery systems and other systems which, while not strictly resource discovery, still have the concept of searching for objects which satisfy some criteria. In this latter category we include systems for name resolution and federated databases. There are important differences between this category and resource discovery, not least the type of queries which are formulated and the type and number of answers which are returned.

As noted in the previous chapter, resource discovery systems take a query and return a number of objects which satisfy the query. These objects need not all be distinct; some may be copies or equivalent objects, but most will be different. In contrast, a name resolution system takes a query (a name) and returns the address to which the name resolves. More than one address may be returned, but these are all taken to be references to the same object, or copies of that object.

Federated databases (Sheth and Larson, 1990) are composed of a collection of cooperating but autonomous databases, possibly heterogeneous. Retrieving data from a federated database requires that a query be directed to the appropriate constituent databases. The task of choosing candidate databases is known as *database selection*, and is a key part of the resource discovery task in a distributed information system.

There are a number of different Internet-based distributed search, information retrieval or name resolution systems, some more widespread than others. These approach the distributed search task in different ways and focus on different aspects of the problem. In addition, the growth of interest in *digital libraries* (whose contents are digital artifacts rather than the physical artifacts held by traditional libraries – paper books and journals, for example) has led to the creation of a number of systems for resource discovery in digital libraries (Dushay et al., 1999; French et al., 1998; French and Viles, 1996; Fuhr, 1999; Liu, 1999).

2.2.1 Domain Name Service

Description

The Domain Name System (Mockapetris, 1987a,b) is the most widespread distributed search system in use on the Internet, and is used to map domain names (eg. `stone.warwick.ac.uk`) onto Internet addresses (eg. `137.205.224.4`). DNS is a replacement for an earlier system of name resolution in which a central database (the `HOSTS.TXT` file) was replicated to all of the machines on the network. The effort involved in propagating updates to this database increased as the square of the number of network hosts, even if the actual resolution of names increased linearly.

Internet domain names are hierarchical, and so DNS breaks the namespace into zones (eg. `.soton.ac.uk`), each of which has a number of nameservers which hold the resource records (the components of the name-address mapping) for the hosts in that zone. When a nameserver is queried about hosts about which it does not hold authoritative data (ie. they are not within its zone), it has three options:

1. give a non-authoritative response from cached data if present,
2. issue a referral to another nameserver which is better placed to answer the query,
3. issue the same query to another nameserver which is better placed to answer the query

The referral information is an expression of the expected knowledge of another nameserver, and is derived from the name hierarchy; a nameserver will typically have pointers to other name servers that can be used to lead to information from any part of the domain tree.

The latter two responses above illustrate two general approaches to distributing queries. In the first (the *iterative* case), the nameserver issues a referral and lets the client pursue the query, whereas in the second (*recursive*) case, the server pursues the query on behalf of the client (ie. the client is effectively delegating the whole of the query task to the server). The difference between these two is further discussed in Section 3.4.

Discussion

DNS is a name resolution system which returns a single address binding for a name, rather than an information retrieval system which returns all records which match a query. The structure of Internet domain names makes it comparatively easy to create an efficient name resolution system, but it is unlikely that this would be the case if DNS were applied to other problem domains.

2.2.2 Harvest

Description

The Harvest Information Retrieval System (Bowman et al., 1995b,a) was originally designed as a general purpose IR tool. Rather than concentrating on the efficient retrieval of information from existing indices, the Harvest project concentrated on the construction of those indices.

In a conventional Web-based indexing system (Koster, 1994) such as Altavista, an autonomous program or *robot* traverses each server which is to be indexed, retrieving each of the objects thereon and adding it to its database. Although there are techniques for minimising the load that this traversal places on the server, it can be a resource intensive operation because under earlier (but still used) versions of the Hypertext Transport Protocol (Berners-Lee et al., 1996) each object retrieval creates a separate TCP connection.

Harvest reduces the network load inherent in remote traversal index building by using software on each site to index the local objects, and then submitting a summary of the objects to the index server. The components of a Harvest system are thus divided into three groups:

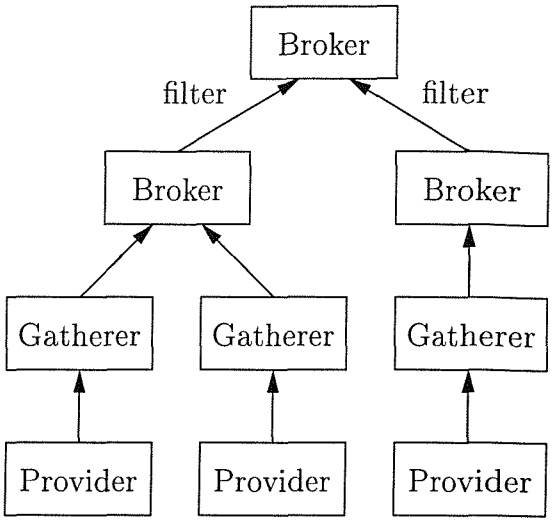


Figure 2.1: The Harvest Architecture

Providers: These are the information servers whose contents are indexed by the Harvest system.

Gatherers: These collect and extract indexing information from the providers.

Brokers: These provide the indexing and a query interface to the gathered information.

The components of a Harvest system are organised as shown in Figure 2.1. The brokers in the system are commonly arranged in a hierarchy, with lower-level brokers passing a filtered version of the summaries that were received from the gatherers to a broker in the level above, which can then provide a refined search capability to users.

Discussion

Harvest does not provide a referral mechanism to guide the client during the resource discovery task; a query sent to a broker can only return references to objects on the providers which the broker's gatherers index. Any global searches over the set of brokers are carried out in a brute force fashion, the query being replicated to each in turn.

2.2.3 CCSO Nameserver

Description

The CCSO Nameserver (Hedburg and Pomes, 1998) (also known as 'Ph') is a white pages database designed to hold a relatively small amount of information about a

large number of entities (eg. an institutional telephone directory), and to provide networked access to that information. These nameservers are local, in that they cannot refer a client to another server which might contain the desired information, although it is possible for a server to contain a list of other servers which can be retrieved by the clients.

Discussion

Ph cannot be used as the basic technology for distributed search, but it is widely used to serve collections of data (in Harvest terms, it acts as a provider). Ph also highlights a common problem with the resource discovery task; if more than one distinct database is to be queried, the databases must either have the same core *schema* (for a field-based system like Ph, the names of possible fields and their uses) or there must be a way of converting one schema into another.

2.2.4 Referral Whois

Description

Whois (Harrenstien et al., 1985), the predecessor to RWhois, is an Internet white pages service containing administrative information such as contact names and address for Internet hosts and domains. The original Whois system consisted of a small number of centralised databases containing all the records in certain top-level domains (eg. one for `.com` and `.org`, one for `.mil` and `.gov`). If a user did not know which database to query (eg. they had a user's name but not their domain), they had to query each Whois server in turn.

RWhois (Williamson et al., 1997) improves upon this by creating a hierarchical namespace for the records, based upon the domain names from DNS. This namespace is broken down into a number of *authority areas*, much like DNS zones, which contain the administrative information for certain types of hosts or domains (for example, the `.uk` authority area would contain information for the subdomains of `.uk`, and for the hosts in those domains).

An RWhois server for a given authority area is able to give authoritative answers for queries about hosts and domains within that area. However, in the event that an authority area is further subdivided (to `.ac.uk`, `.org.uk` and `.co.uk`, for example), the server for the parent area is not able to give authoritative answers for queries about hosts and domains in the subareas; the subareas have their own RWhois servers which answer authoritatively.

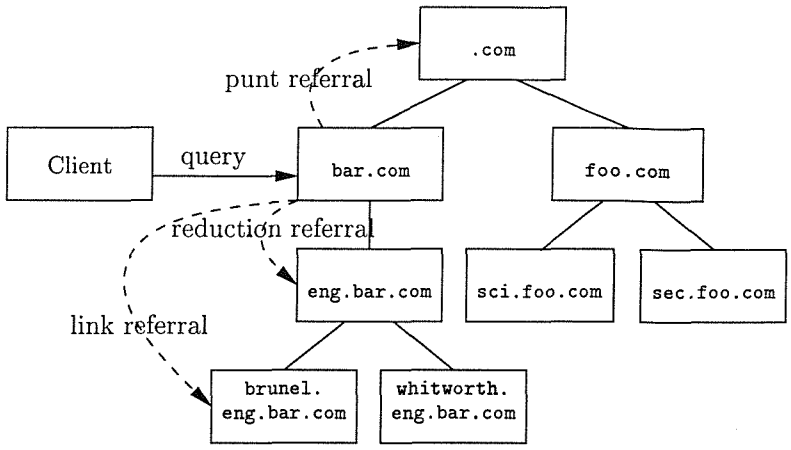


Figure 2.2: RWhois referrals

Thus, RWhois servers may issue referrals to direct the client to another server (*query routing*). RWhois identifies three distinct types of referral:

link: The authority area of the referred-to server is the same as that in the original query (ie. the referral is to a server who will be able to authoritatively answer the query).

reduction: The authority area is not equal to the query, but this referral is still constraining the search space.

punt: No authority is sent; the client is being referred up the hierarchy and the search expanded.

In Figure 2.2, the client is issuing a query about the host `brunel.eng.bar.com` to the RWhois server with authority area `bar.com`. If the server issues a referral to the server with authority area `brunel.eng.bar.com`, this is a *link referral*, whereas a referral to the server with authority area `eng.bar.com` would be a *reduction referral*. If the query was not within the authority area of the initial server (for example, a query about the `sci.foo.com` domain), a *punt referral* would be issued to the server's parent, whose authority area is `.com`.

Discussion

The structured white pages records which RWhois is designed to search for could be modified to contain other types of data, such as bibliographic data. However, the type of knowledge that RWhois servers have about each other is closely linked to the hierarchy of authority areas, which places restrictions on the type of data each server may hold. It would be possible to partition the search space into authority areas based on a hierarchical subject classification like Dewey Decimal, but would be

impractical to do likewise based on a keyword subject classification – this restriction may make RWhois unsuitable for some applications.

2.2.5 Whois++

Description

The WHOIS++ (Weider et al., 1996; Faltstrom et al., 1998) directory service is a different refinement of the Whois network white pages service and is intended to provide a simple, extensible white pages directory service using a template-based information model and a flexible query model. While not designed to be a general tool for distributed information retrieval, it addresses many of the problems which affect such systems.

Like the RWhois system, WHOIS++ uses a system of referrals to direct the client from one server to a more relevant server and constrain the search space. If a query is formulated which does not specify any information pertaining to the namespace, RWhois attempts a global search - an expensive proposition. WHOIS++ does not restrict the referrals to a simple hierarchy; but generates them from a mesh of *forward knowledge* which describes the contents of servers (in much the same way as Harvest summaries).

This forward knowledge comes in the form of *centroids*, partially instantiated records which subsume some group of records. These centroids may be constructed for any database schema; in WHOIS++ , the fields in a centroid contain a list of all of the words which appear in the fields of the records which the centroid represents. For example, a database containing three records, each of which has a contact name field with respective values 'John Smith', 'Peter Jones' and 'Robert Smith', would generate a centroid whose contact name field has the value 'John Smith Peter Jones Robert'. This 'words-appearing-in' summarisation is performed independantly on each field of the records. Although the WHOIS++ specification does not require the elimination of stop words or stemming of terms, these techniques could easily be used to improve the effectiveness of centroids.

In the WHOIS++ system, the individual white pages records are held by a layer of *base level servers*. The contents of these are summarised by a layer of *indexing servers* which hold forward knowledge about the base level servers. In turn, these may be described by the forward knowledge held by a second layer of indexing servers, and so on. Unlike RWhois, there need not be a strong correspondence between the domain name hierarchy and the records contained within a given server.

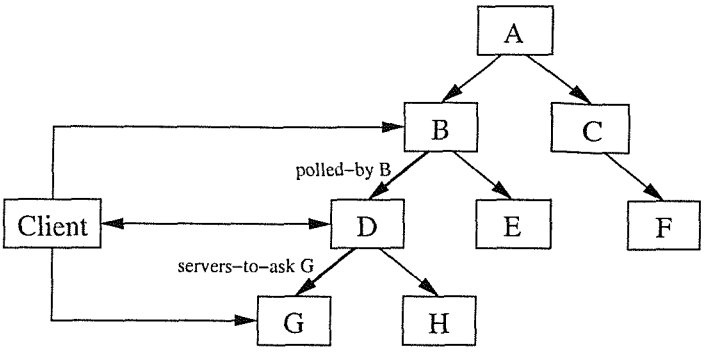


Figure 2.3: WHOIS++ referrals

The flexibility of the centroid and mesh approach of the WHOIS++ system lends itself to a rich range of server interactions (Faltstrom et al., 1996). A WHOIS++ server may issue two types of referral. The first, a *servers-to-ask* referral, is issued when a server contains centroids for another server's contents which are relevant to the query, and acts to constrain the search to more relevant servers. The second type, the *polled-by* referral, indicates that a centroid for the current server is held by a different server (which may hold relevant centroids for other servers), and expands the search.

Discussion

WHOIS++ is arguably a more flexible approach to a distributed white pages directory service than RWhois, but is not as widely used. Even if WHOIS++ has been largely abandoned, elements of it have found use in other protocols; the Common Indexing Protocol (see Section 2.2.7) is a further development of the index building techniques in WHOIS++ , while later developments of RWhois use similar types of forward knowledge. There are some notable omissions from the WHOIS++ specification. The RFCs do not explicitly state how a query is to be matched against a centroid; it is not apparent whether all the words in a query phrase should be present in a centroid for a referral to be generated.

Together with the RWhois system, WHOIS++ is the origin of the general idea of query routing which this thesis explores.

2.2.6 ROADS

Description

ROADS (Resource Organisation and Discovery in Subject-based Services) (Knight and Hamilton, 1996) is a project in the JISC-funded Electronic Libraries (eLib)

programme. It uses WHOIS++ as the basis for a system which could be used to build *subject gateways*, human-compiled metadata repositories for Internet resources. The aim of the subject gateways is to provide a high-quality alternative to existing WWW search engines (Altavista, Lycos) and resource directories (Yahoo).

The gateways were organised by subject classification, with separate gateways for medicine, business studies, history and so on. The role of ROADS in this was to provide a means whereby queries of an interdisciplinary nature could be redirected from one gateway to another, hence the use of WHOIS++ .

Discussion

ROADS is primarily of interest because it uses WHOIS++ for something other than white pages directory information. ROADS uses IAFA templates (see Section 2.4.2) to store details about the resources described by the subject gateways, although it can translate these to a number of other formats (Dublin Core (DCMI, 1999), LDIF (University of Michigan, 1996) and SOIF (Hardy et al., 1996)).

By virtue of a concrete implementation, ROADS clarifies some of the ambiguities in the WHOIS++ specifications, most notably the matching of queries to centroids. ROADS requires that all the words in a query phrase must be present in a centroid for a referral to be generated to the server whose centroid it is.

2.2.7 Common Indexing Protocol

Description

The Common Indexing Protocol (Allen, 1997) is a further development of the distributed indexing techniques introduced by WHOIS++ . CIP is not a protocol for data access or information retrieval, but is used to pass indexing information between servers to facilitate query routing (ie. by issuing referrals) and make future data accesses by clients more efficient. CIP must therefore be used in concert with a data access protocol, such as Z39.50 (Z39.50 Maintenance Agency, 1995) or HTTP (Fielding et al., 1999), which is responsible for issuing the referrals that are generated from the indices built by CIP.

The basic premise of index passing is that an index object generated by lossy compression methods (such as those used by WHOIS++) still contains useful hints for routing queries. As servers collect index objects, they may choose to remove the redundancy between those objects by aggregating them into one. The index objects used in CIP are more sophisticated than the centroids found in WHOIS++

; not all queries will benefit from the same type of index, so new types of index object may be created (using a common syntax based on the MIME specification for structured Internet mail (Allen and Mealling, 1998). This introduces a problem related to the schema conversion problem (of CCSO and others); a server may not understand the index objects it receives. Such a heterogeneous network of servers is advised against in CIP, though there may be situations in which it is necessary.

CIP identifies two modes in which index objects are passed between servers, *index polling* and *index pushing*:

Index polling: This is a symmetric relationship between two servers which have agreed to share index data. Both sides may initiate the dialogue; the polling server may request an update of the polled server's index object, or the polled server may notify the other if its index is modified.

Index pushing: In this mode, a server simply sends an index object to another server which may then handle it as it pleases. This mode is intended for leaf nodes which only want to pass their index objects to a higher level of the mesh (cf. Harvest providers).

Discussion

CIP is a key specification on the IETF standard track, and promises a degree of integration between other Internet technologies; later versions (v2.0 or higher) of the Referral Whois protocol (Blacka et al., 1998) work with CIP meshes as well as its own hierarchical referrals, and indices can be generated for LDAP (Wahl et al., 1997) directories.

2.2.8 X.500 and LDAP

Description

X.500 (ITU, 1993a) is a distributed directory system (serving white pages information) which forms part of the ITU Open Systems Interconnection family of standards. The directory provides a lookup facility by which OSI objects can be located given only their name. Each object in the directory is described by a set of attributes whose values form part of the name used to specify that object.

X.500 assumes that the directory information is organised as a tree (the *Directory Information Tree*), and that servers (*Directory System Agents*) provide access to sections of the tree (*Naming Contexts*) to the clients (*Directory User Agents*). The entries stored in the DIT are field based records consisting of a set of attributes

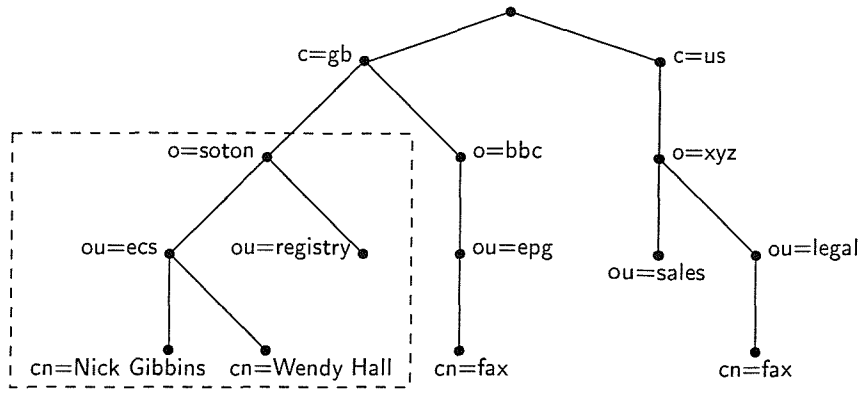


Figure 2.4: The objects in the hypothetical DIT shown above are labelled with their RDNs, which may be concatenated to form the DN for an object. A DSA whose naming context encompassed the subtree below `o=soton` would have the context prefix `o=soton,c=gb`.

which are constrained by a schema. These schemas are described in *subschema entries* held by each DSA. X.500 specifies a *Directory Access Protocol* which binds the DUAs to the DSAs.

There are two types of name used to identify objects in the DIT. The first, *distinguished names*, are part of X.500 while the second, *descriptive names* are a concept introduced by Neufeld (1989).

Within each entry in the DIT, one or more attribute values may be marked as *distinguished*. The set of the distinguished attributes on an entry are known as the *relative distinguished name* of the entry, and must be unique amongst the entry's siblings. The *distinguished name* for an entry gives its location in the DIT by specifying a path from the root of the tree to the entry; the DN is the sequence of RDNs for the entries on that path. Within such a strict hierarchy, it is often not possible to determine a single place to put an object, so it is common to insert aliases for an object in different places in the DIT; these aliases contains a reference to the DN of the objects they stand for.

Because the directory is distributed, it may not be possible for one DSA to resolve a given distinguished name. In this case, the DSA uses the DNs of the root entries in other naming contexts (the *context prefix*) to identify the DSA which is best placed to handle the request. Each DSA has a set of *references* which describe how its naming context fits into the DIT as a whole by specifying ancestor/child/cross-reference relationships with the naming contexts of other DSAs. It is this set of references which is used to generate referrals to other DSAs.

In summary, distinguished names are hierarchical and the distribution of the X.500 naming service relies explicitly on this hierarchy. The hierarchy in the tree is similar to that used by DNS and divides the space on geographical and organisational lines (a sample DN for a personal entry might be `cn=Nick Gibbins, ou=ECS, o=University of Southampton, c=gb`).

If distinguished names identify objects by specifying the path taken to reach them, descriptive names identify objects by listing the attribute values they must have (known as *naming attributes*). The order of these attribute values is unimportant, and the set of attribute values need only be populated enough to unambiguously identify the object.

Descriptive naming therefore exists as a naming technique which does not rely on the name hierarchy of the DIT for resolution. Also, because there may be more than one descriptive name for an object, it is no longer necessary to populate the DIT with alias entries to provide alternate names for objects. In order to resolve descriptive names without resorting to global search, objects may have a unique *registered name* (unique amongst the descendants of the immediate parent of the object) which must form part of the descriptive name of all objects beneath it. Although superficially similar to a context prefix, a registered name is a set, not a sequence, of RDNs.

LDAP (Wahl et al., 1997) is a *Lightweight Directory Access Protocol* which provides the DUA to DSA binding directly over TCP without much of the session/presentation overhead of X.500 DAP. Later versions of LDAP (v3 and higher) allow DSAs to send referrals to DUAs; earlier versions (Yeong et al., 1995, 1993) required the DSA to handle referrals itself (performing a recursive search, in DNS terminology) without resorting to the return of such referrals to the DUA.

A related development, the *Directory Assistance Service* (Rose, 1991), provides a different interface to X.500 DSAs by splitting the DUA functionality and interposing a *Directory Assistant* between a simplified DUA (the *DA-client*) and the DSA. The DA consists of two entities, a *DAP-listener* which speaks X.500 DAP to the DSA, and a *DA-server* which uses a simpler protocol, the *DA-protocol*, to speak to the DA-client. This division hides the handling of referrals from the user client in a similar fashion to the earlier versions of LDAP.

Discussion

The OSI directory is a powerful model of distributed search which has already found use in studies in domains other than white pages lookup (Barker (1992) gives

a description of an X.500-based system for accessing bibliographic information). Although X.500 is a mature standard, it is not widely used, in part due to the complexity of implementation and a lack of political will to create the infrastructure for the DIT. The chief weakness of the OSI directory lies in the rigid naming hierarchy which all objects must use. Alternate naming solutions such as descriptive naming go some way to providing a more flexible naming system, but still rely on the DIT for their distribution.

Other X.500-related standards such as LDAP have found wider acceptance, and have been used in systems for general resource discovery (Roszkowski and Lukas, 1998) and hypertext link resolution (DeRoure et al., 2000).

2.2.9 Nomenclator

Description

The Internet Nomenclator Project (Ordille, 1998) aims to integrate publicly available CCSO servers into a tree reminiscent of the X.500 DIT and allow searching across these servers, even though they may have differing database schemas. The Nomenclator server takes a data fusion approach and provides a translation from the schemas used by existing data repositories and CCSO servers (the *local view*) to a global schema understood by Nomenclator clients (the *world view*).

Nomenclator adopts a descriptive naming approach (Ordille and Miller, 1993), using referrals to successively constrain the search. It uses cached responses and its knowledge of the translations between the world view and the local views to ensure that a query expression is only passed on to those CCSO servers for which the query is relevant. The Nomenclator system is thus divided into two groups of components:

Distributed Catalog Service: This gathers metadata about the data repositories, such as schema types, attribute value constraints, translation techniques and known patterns of data distribution across the repositories.

Query Resolvers: These use the metadata in the DCS to direct user queries to appropriate data repositories. There may be several different resolvers in the system (eg. for different organisations).

The referral mechanism resides in the DCS; *catalog functions* return a list of references to *data access functions* (which can tell a resolver how to query specific data repositories) or to other catalog functions and a template which describes the scope of those functions. A list of the relevant data access functions is generated by

matching the referral templates against the query expression, in a similar way to the matching of WHOIS++ centroids against queries. User clients talk to the query resolvers using a common query protocol, the Simple Nomenclator Query Protocol (Elliott and Ordille, 1998); there is no direct communication between the clients and the DCS.

Discussion

Nomenclator differs from WHOIS++ in that the client does not directly query the database servers, relying instead on the intermediate Query Resolver; the referrals are contained within the server side of Nomenclator. Although the distributed catalogue system may be distributed (as is suggested by its name), no specific technique for its distribution, in particular for searching or data integrity, is given.

The generation of Nomenclator referrals is governed by a set of rules which, summarised, require that a referral is only generated when a query is completely covered by (more specific than) the template on a catalog function, and that the templates used to generate specific referrals are in strictly increasing specificity. The latter requirement prohibits the generation of referrals which increase the search space; all referrals narrow the search.

2.2.10 Z39.50

Description

Z39.50 (Z39.50 Maintenance Agency, 1995) is an ANSI standard for an interoperability protocol which allows clients to search a variety of databases. Originally designed for use in libraries, it primarily deals with schemas for bibliographic data (such as MARC records - see Section 2.4.1). Z39.50 allows a user to use a single application to search multiple heterogeneous databases by using a common protocol between all clients and servers (in Z39.50 terminology, *Origins* and *Targets*) and by standardising the structure and semantics of the search query.

Discussion

As it stands, Z39.50 does not address distribution issues, but does represent one approach to the problem of schema translation, namely by adopting a common profile which provides a core set of functionality. The Bath Profile (Lunau et al., 2000) is one such profile for Z39.50, designed for library functions such as the search

and retrieval of bibliographic records for inter-library loans or the construction of union catalogues, and defines types of query which must be supported in addition to schema definitions. Z39.50 shows some promise for query routing-based distributed search, as the work on ZBroker by Lin et al. (1999) demonstrates.

2.2.11 Uniform Resource Name Resolution

Description

Uniform Resource Names (Moats, 1997) are a development of the Uniform Resource Locator scheme (Berners-Lee et al., 1994b) used to reference Internet information resources (Web documents, etc). Although they are not yet in common use, they promise to provide a more sophisticated and robust namespace than is available at present. Strictly speaking, URLs are addresses and not names, because they encode the physical location of a resource as a machine name (here we overlook the use of DNS tricks which allow a domain name to resolve to more than one IP address). In contrast, URNs are names and not addresses, such as URNs for International Standard Book Numbers, of which `uri:isbn:0123456789` is an example.

The translation of URN names to URL addresses involves an indirection mechanism which also enables the transparent mirroring and caching of resources. This indirection requires a resolution service, and there have been several proposed. Some of these resolution services harness existing technologies such as HTTP (Daniel, 1997) or DNS (Daniel and Mealling, 1997), but the problem of finding a suitable resolver for a URN remains.

The Resolver Discovery Service (Sollins, 1998) proposes a three tier model for URN resolution. At the bottom lie the URN resolvers which map URNs onto URLs. Above that lies a mesh of RDS servers which accept a URN and return either a reference to a URN resolver or a rule which generates a reference to another RDS server. The top tier is occupied by a Global NID (Namespace ID) Registry which is used to identify the first RDS server to be contacted. This model, though complex, should allow the partitioning of the RDS database on boundaries other than those of the name delegation denoted by the URN namespaces.

Discussion

The RDS takes a similar approach to name resolution to X.500 and DNS. The search space is hierarchically partitioned with a server having authority over each partition. However, the use of a global registry to identify which server to talk to

initially sets RDS apart from X.500 and DNS, which allow queries to be started anywhere in the server tree. The RDS global registry is effectively a root server which must be queried every time a URN is resolved; X.500 and DNS both allow referrals which point towards the root of the server tree, so queries need not be started at the root.

For example, given a query (or name to be resolved) of `uri:isbn:0123456789`, an RDS client would consult the global registry to find an initial resolver to query (which would be the top-level resolver for the `isbn` type). If the query were initially presented to a different resolver (for example, one for US patents), resolution would fail because there would be no way to generate a referral to the global registry (and from there to the appropriate resolvers for ISBNs).

2.2.12 STARTS

Description

STARTS (Gravano et al., 1997) is a Stanford proposal for a protocol and metadata schema to be used by meta-search engines that take a query and submit it to a number of other search engines (examples are MetaCrawler or Dogpile). The merging of returns from the queried sources is difficult because they may be very different: they may not determine relevance in the same way, nor use the same ranking algorithms, nor even the same query language.

Discussion

STARTS informs resource discovery by providing a way for a meta-search engine to assess the capabilities of other sources, but is not a technique for distributed search as such, rather a component of a heterogeneous distributed search system.

2.2.13 Napster

Description

Napster is a system designed to enable Internet users to share `.mp3` files (MPEG Layer 3 digital audio) amongst themselves. Although it has attracted a lot of attention (and not a few lawsuits) due to the uses to which it has been put and their implications with respect to copyright and fair use, it remains a good example of a distributed search system.

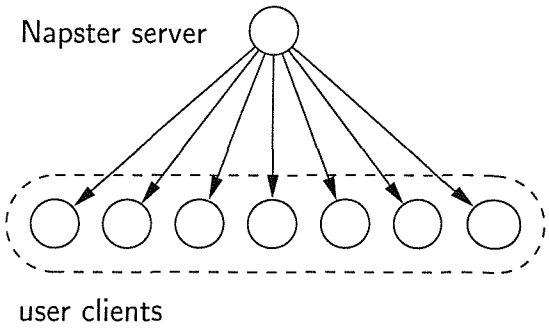


Figure 2.5: The Napster Architecture

All Napster users run a piece of software which is simultaneously a search client and a server. When the Napster software is started, it builds a catalogue of the user's files (or rather, those of the user's files which they wish to share) and sends this description to a central index server. When a Napster user wishes to find a particular file, the client sends a query to the central index server which compares the query to its collection of catalogues and returns a list of the locations of likely hits, these locations being other Napster clients. The client then attempts to fetch the file from these locations, treating the remote Napster clients as file servers.

Discussion

Although it has proved phenomenally successful, the design of Napster is flawed in that it contains a single point of failure, namely the central index server, without which the system cannot work. This weakness has been aptly demonstrated both by the actions of a number of US universities which now block access to the Napster server with their firewall in order to reduce the network traffic that Napster generates, and by a recent court ruling ordering the closure of the index server. The index server also bears a disproportionate part of the system load, because all client interactions are mediated through it.

While the Napster service was still in operation, the issue of the server bottleneck was addressed by providing a number of different servers and letting the user choose which server their client connected to. However, there was no communication between these servers, so `.mp3s` known to one server would not be known by another. This rather defeated the purpose of distributing the server functionality, because users would either pick the 'best' server (being the one which knew about the most `.mp3s`, and so the one which had to support the greatest number of client connections) or would query each one of the servers in turn.

2.2.14 Gnutella

Description

Gnutella is a system with similar origins to those of Napster, although it enables the sharing of more than just `.mp3` files and has a markedly different architecture. In Gnutella there is no central index server; instead, queries are broadcast to the network at large.

When a Gnutella server joins the network, it sends out a flood message to discover the other servers on the network. All flood messages in the system contain a globally unique ID which should be shared with no other message and a *time-to-live* field (TTL) which limits the distance in hops it can be sent from the originating server. When a server receives a flood message, it records in its routing table the GUID of the message and the name of the server from which it directly received it, and forwards the message with a decreased TTL. Later messages containing the same GUID are silently discarded, as are messages with a TTL less than one.

The responses from the discovered servers contain information about the number of files owned by a server and the total size of those files. When a server sends a response, it looks up the GUID of the message to which it is responding in its routing table and sends the reply to the server from which it received the original message. Other servers receiving responses behave similarly, so a response traces a path which is the reverse of the path taken by the original message.

This is illustrated in Figure 2.6; the central grey node is the user's client, while the grey node toward the lower left corner is the server which contains the goal. The edges (both solid and dotted) indicate mutual awareness between servers that has been gathered by means of the flood messages sent when servers enter the system.

When a user wishes to locate a particular file, their Gnutella server floods the query in a similar way to the discovery messages. The query responses contain the location of the matching files, which are then fetched by the user's server using HTTP. The solid lines in Figure 2.6 show the edges over which the query is first sent during query flooding, and the numbers within the nodes give the shortest distance from the originating node (in effect, the initial time-to-live minus the current time-to-live at that node). The dashed line around the central connected component of the network shows the flooding boundary brought about by the chosen initial time-to-live (in the figure, five).

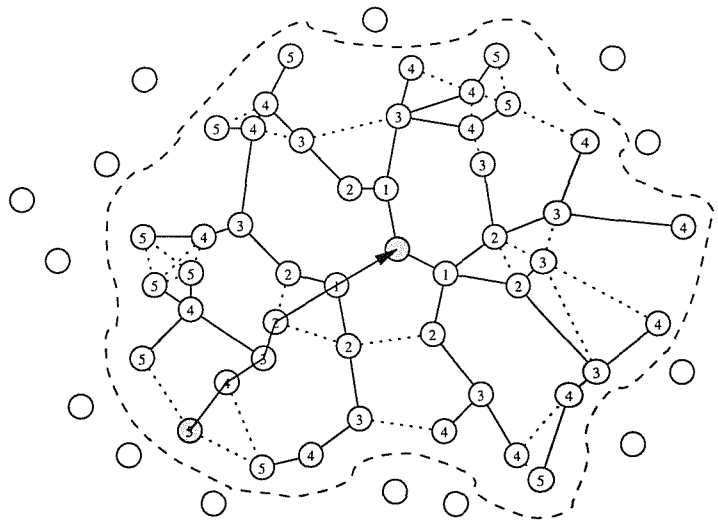


Figure 2.6: Search flooding in Gnutella

This query flooding induces a spanning tree over the network with the originating client at its root; returning search results to the client is a matter of tracing the (unique) path from the relevant server to the root.

Discussion

Gnutella is truly distributed and so does not have the problem of a central index server that Napster has, but does have problems of its own. Chief amongst these is the question of message flooding, which is generally considered wasteful. The scalability of the Gnutella network became an important issue when the Napster service was served with an injunction on 26 July 2000 and large numbers of its users switched to using Gnutella instead (the 'Napster Flood'). This influx of users accompanied a severe increase in the time taken to process Gnutella queries (an informal analysis of this is given by Ritter (2001), one of the Napster developers).

Some of the criticisms directed at Gnutella over this issue are purely implementation dependent; each server sends a discovery message every minute or so, which constitutes a substantial overhead. Some informal studies by the Gnutella designers estimate that as many as 50% of messages sent by the system are related to the initial server discovery on joining the system, and that this could be dramatically cut if each server were to keep a record of the information contained in response messages that it forwards (in effect, caching results closer to the originating client).

Implementation issues aside, the flooding of messages means that messages are frequently sent to servers which contain no relevant files, and the ad-hoc network topology of the system leads to the presence of cycles which cause the same message

to be received by a given server several times. The TTL imposed on messages also leads to a horizon effect whereby a query flooded into the network may not reach a server which contains relevant files because the server is out of range, directly affecting the effectiveness of the search. This issue could be addressed by increasing the initial TTL and rebroadcasting the query from the original starting point (an expensive proposition), by rebroadcasting the query from a different starting point close to the horizon of the original search (still not guaranteed to produce an answer) or by modifying the topology of the network in order to reduce its diameter to less than the TTL. Of these approaches, only the latter has been attempted so far, by creating a backbone network of servers which connect distant parts of the network and reduce its overall diameter.

2.2.15 Freenet

Description

Freenet (Clarke, 1999) is another system designed to enable users to share files, but imposes further constraints on its operation than either Napster or Gnutella. In addition to decentralisation, the design for Freenet called for anonymous publication and retrieval and the duplication of popular material.

Strictly speaking, Freenet is a name resolution system rather than a distributed search system, since it is designed to be able to retrieve documents with fixed, predetermined keys (names). Each server in the system maintains a data store containing documents with their associated keys and the address of another server known to hold the same document. A server may also hold records about documents which it has deleted, but which it knows other servers hold.

When a Freenet user issues a query, the user's client sends a message containing the key specified to a server, usually one local to the user. The server compares the key with the keys for documents it holds and returns a matching document if one exists. If no copy of the document exists on that server (no key exactly matches the query), it finds the closest matching key to the query and forwards the message to the server associated with that key.

If a matching key is found, the document it references is returned to the client by the reverse path, otherwise the request is forwarded again in the same manner. The document may also be cached by the servers on the route home in order to facilitate future requests for the same document. If the message reaches its maximum range (its TTL reaches zero), arrives at a server for a second time or finds that the server to

which it is being forwarded is unavailable, the most recent server to have forwarded the message resends it to the server that is associated with the next closest key in its data store, so backtracking on failure.

Discussion

The immediate advantage of Freenet over Gnutella is that it conducts searches in a depth first manner, reducing the immediate impact on the network. Also, the ability to locally cache documents makes the Freenet network (as defined by servers' knowledge about each other) adapt to changing user demand.

However, Freenet as implemented is unusable as a resource discovery system because the keys by which documents are identified are cryptographic hashes either of a keyword or of the document itself; the system can only be used to retrieve documents whose keys are known a priori. Another drawback is that the search strategy employed by Freenet will only return the first matching document. This is not a problem in a name resolution system where all the object identified by a name are equivalent, but is not appropriate for use as a resource discovery system where a user expects to receive several matching documents.

2.3 Agent Technologies

Agent-based computing is widely held to be a software engineering paradigm of growing importance (Sargent, 1992; Maes, 1994). However, although the term *agent* is now commonly accepted in the fields of computer science and artificial intelligence, it has a plethora of subtly differing definitions.

In their oft-cited 1995 paper (Wooldridge and Jennings, 1995), Wooldridge and Jennings identify two main types of agenthood and list the characteristics necessary for each. The first, *weak agency* requires the following four properties:

autonomy: the ability to operate without the direct intervention of humans or others

social ability: the ability to interact with other agents

reactivity: the ability to perceive their environment and respond to changes in a timely fashion

proactivity: the ability to take the initiative and display goal directed behaviour

Strong agency, the second type, is a subset of weak agency in which the agent's state is characterised using mentalistic notions such as belief, knowledge, intention

or obligation and its communications using speech act theory (Searle, 1969), in addition to fulfilling the criteria for weak agency. This *intentional stance* (Dennett, 1971) is important, since it allows us to reason about the behaviours of complex Multi-Agent Systems using naïve psychological terms. In his paper on Agent-Oriented Programming (Shoham, 1993), Shoham summarises this as:

> An agent is an entity whose state is viewed as consisting of mental components such as beliefs, capabilities, choices and commitments. These components are defined in a precise fashion, and stand in rough correspondence to their common sense counterparts.

A very pragmatic definition of agency has been proposed by Huhns and Singh (1997), as follows:

> A system containing one or more reputed agents should change substantively if another reputed agent is added to the system.

There are three main components to a system for agent-based computing:

- a language for determining the behaviour of agents, or *agent programming language*
- a language for representing the knowledge of an agent, or *content language*
- a language for expressing the messages sent by agents, or *agent communication language* (ACL)

The first is an implementation of an *agent theory*, which Wooldridge and Jennings describe in (Wooldridge and Jennings, 1995) as a specification for an agent. An agent theory is an attempt to represent the properties of an agent, and as such affects the internal structure, behavioural characteristics and implementation of an agent. There are a number of agent theories at present, including Cohen and Levesque's theory of intention (Cohen and Levesque, 1990a), Moore's theory of knowledge and action (Moore, 1990), Rao and Georgeff's Belief, Desire, Intention (BDI) architecture (Rao and Georgeff, 1991) and others beside.

While the agent programming languages specify and constrain the behaviour of the agents, and the content languages are used to express the beliefs held by the agents, the agent communication language provides the definitions of the messages sent between agents. There is a strong relationship between the three languages; the content language affects both the agent theory (by specifying an ontology with which the agent must reason) and the agent communication language (an agent's beliefs are commonly the subject of inter-agent communication).

In addition to these three languages, an agent system also requires a number of ancillary services by virtue of its nature as a loosely coupled distributed system. By loosely coupled, we mean that the interactions between the entities in the system (the agents) are not constrained by design-time decisions, but that the agents can interact and form relationships in an ad-hoc and opportunistic fashion. These ancillary services include, but are not limited to, the following list (adapted from (Gasser, 2000)).

- certification services, which enable an agent to verify the origin of another agent
- security services, which enforce the social norms necessary for a functioning system, such as trust, veracity or data integrity
- resource description and discovery services, which enable agents to offer services or to discover other agents which can provide certain services
- economics services, which provide facilities for charging and managing economic interactions between agents.

These services all support the interactions of the agents in an agent system, but are not considered to be necessary prerequisites for agency, nor are they encountered exclusively in agent systems. Indeed, these ancillary services are commonly encountered features of loosely coupled distributed systems in general, particularly where the system components are not automatically trusted (so raising the requirement for certification and security services). Similarly, if a system component is to be able to make use of other components whose existence was not known when it was being designed, some form of resource or service discovery must be a requirement. Examples of services in these areas can be found in the CORBA Services specification (CORBA, 1995), which describes a level of infrastructure which lies above the transport-oriented (or communication-oriented) infrastructure of the core CORBA specification.

Of these services, we are most interested in the provision of resource description and discovery services in agent systems (as the subject of this study).

2.3.1 The DARPA Knowledge Sharing Effort

The DARPA Knowledge Sharing Effort (KSE) (Patil et al., 1992) was an early effort to produce a standard framework for agent communication, and has met with some success. The main deliverables of this work were the Knowledge Query and Manipulation Language (KQML) (KAG, 1992; Labrou and Finin, 1997), an

agent communication language, and the Knowledge Interchange Format (KIF) (Genesereth and Fikes, 1992), a content language.

KQML provides a rich set of speech acts for agent communication with an s-expression syntax, and has been used in a wide variety of projects: Agent-K (Davies and Edwards, 1994) and AgentBuilder (AgentBuilder, 1998) use KQML in a framework based on Shoham's AOP, while other systems such as Stanford's Infomaster (Genesereth et al., 1997) use it with KIF as an interlingua to express queries from heterogeneous sources expressed in SQL or other query languages.

The linkage between KQML, KIF and an agent programming language is less strong than might be expected. Cohen and Levesque (1990b) note that there are no formal semantics given for the KQML performatives, so their meaning (the illocutionary effect upon the recipient) is unclear. The independence of KQML from the content language KIF means that self-defeating speech acts could be sent from one agent to another (eg. an agent could send a message expressing Moore's paradox, "I hereby inform you that p is true and that I do not believe that p is true") because the message content cannot be checked for compatibility with the performative type.

The set of speech acts provided by KQML is incomplete. Although several directives exist (eg. `ask-if` or `subscribe`), the most fundamental directive, *request* is not present. This means that an agent cannot ask another to perform an arbitrary action, but must instead use the `achieve` directive to ask it to make the postcondition of the action true. Since all the other directives may be expressed in terms of *request* (eg. `ask-if` is equivalent to requesting someone to tell you if a sentence is true), this omission is an important oversight.

Similarly, Cohen and Levesque also note that KQML does not include any *commissive* speech acts (those which would commit an agent to a particular course of action); KQML agents cannot accept proposals, promise to perform tasks or agree on a matter under consideration. KQML is designed to be extensible, so these speech acts could be added to the set, but this requires KQML developers to be aware of these new acts and their required behaviours.

KIF is a declarative language for the representation of knowledge by computer programs and was one of the deliverables of the DARPA Knowledge Sharing Effort, along with KQML. Strictly speaking, KIF is an *interlingua*, a language for communicating knowledge between computer programs, but not necessarily used internally by them (although it can well be used for this purpose).

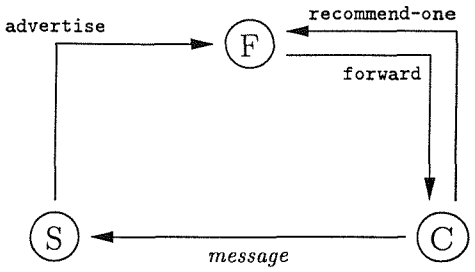


Figure 2.7: KQML Facilitator

KIF has a simple s-expression (Lisp-like) syntax and is based on first order predicate logic. This expressivity is important; KIF can express sentences which relational database languages and languages based on Horn Clauses alone (eg. Prolog) cannot. KIF's suitability as an interlingua has led to its adoption by several projects as a neutral intermediary for expressing queries in different database schemas. The Infomaster system (Genesereth et al., 1997) from Stanford University and the InfoSleuth system (Bayardo et al., 1996) from MCC both use KIF to translate queries in a heterogeneous database environment, while the TAMBIS project (Baker et al., 1995) at the University of Manchester takes a similar approach with a different ontology language, GRAIL.

The DARPA KSE provides a number of speech acts within KQML which allow agents to discover which agents provide certain services. In Figure 2.7 is illustrated a simple system consisting of three agents: a server agent (S) which is providing some service, a client agent (C) which is trying to find an agent which can provide that service and a facilitator agent (F) which matches clients to servers. The server begins the exchange by sending an `advertise` message to the facilitator which contains a message template which will match messages that the server can process. When a client wishes to find an agent which can provide a service, it composes a message which would invoke that service (leaving the `:to` field empty) and sends that message as the body of a `recommend-one` agent to the facilitator (as with many KQML performatives, `recommend` is available in `-one` and `-all` variants depending on whether the querent wants one answer, or an exhaustive list of all answers). The facilitator responds by sending a `forward` message which contains the server's advertisement, which provides the client with the knowledge necessary to be able to invoke the desired service on the server.

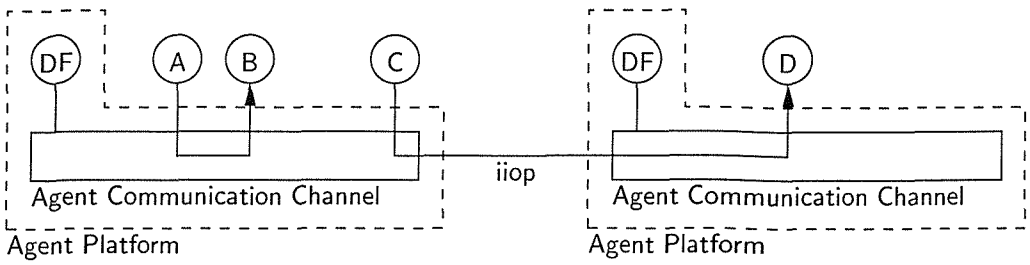


Figure 2.8: The FIPA Agent Platform

2.3.2 FIPA

The Foundation for Intelligent Physical Agents is a non-profit association which aims to increase interoperability between agent systems through a set of specifications for an agent architecture, including a content language and ACL (FIPA, 1997b). FIPA have taken a stance of strong agency, and so the FIPA ACL is similar to KQML in many ways. It presents a rich set of speech acts and has an s-expression syntax, but addresses the problems with KQML discussed above.

A notable facility in the FIPA ACL which adds a level of abstraction to the above are its *protocols*, characteristic exchanges of messages, used for common conversation like requests, auctions or contract nets. The use of protocols in FIPA allows the agents to better reason about their conversations. From a programmer's point of view, protocols provide a shortcut into agent communications which allows sophisticated behaviour without excessive attention to detail.

The FIPA content language *sl*, standing for semantic language, is a first order modal logic with identity and so is more expressive than KIF. sl is used to define the communicative acts in the ACL, giving them the semantic grounding that KQML lacks. The FIPA Specification includes far more than the ACL and sl, defining a standard CORBA-like environment (OMG, 1996) in which the agents operate. This environment or *agent platform* (AP) consists chiefly of the *agent communication channel* (ACC), a bus which agents use to communicate with each other in much the same way as the *object request broker* or ORB mediates inter-object communications in CORBA. Communication between agents on different ACCs is also handled by the ACCs, which communicate with each other by means of the CORBA Internet Inter-ORB Protocol (IIOP).

In Figure 2.8, the agents labelled A and B can communicate via their local ACC because they are both on the same AP, whereas a message sent from agent C to agent D first goes to C's local ACC, then to D's ACC (via IIOP) which delivers it to its destination.

FIPA also specifies a number of standard agent management services (corresponding to the ancillary agent services in Section 2.3) which are deployed as part of the agent platform, of which the *directory facilitator* (DF) is a key example. The directory facilitator is an agent which provides a yellow pages service (i.e. describing agent services or capabilities, equivalent to the resource description and discovery services from Section 2.3) to the other agents in the system. The FIPA DF service is a simple one, enabling agents to register (or deregister) the services which they provide and to search for agents which can provide given services, and exists on every agent platform.

A key difference to note between the FIPA DF and the equivalent service in the DARPA KSE model is that the DF functionality is implemented as a set of services which are opaque expressions from the ACL layer, where the KSE implements its service discovery functionality as speech acts within KQML. Consequently, the FIPA DF can be considered to exist at a level of abstraction above that of the KQML solution. This decision was taken for pragmatic reasons; by keeping facilitation separate from the agent communication language, the specification of the ACL is kept simple because it does not need to model the capabilities of agents.

2.4 Bibliographic Metadata

Currently, the majority of systems for searching the Internet (including the centralised systems such as AltaVista or Lycos) use full text searching techniques. This type of search has been known to be flawed for some time, but it has only been with the emergence of indices containing millions of full text records for global distributed information systems such as the Internet that we are seeing the full extent of their shortcomings. A search for a common word or phrase in such a system may yield hundreds of thousands of hits, almost all of which are of no relevance to the user who made the query.

The move has been made towards attaching some semantic information to electronic resources in order to make information retrieval more useful by reducing the number of these false hits. This semantic information is metadata, information which describes some information resource, commonly including the item's author, publisher, date of publications, classification number or edition.

It is unsurprising that the design of the metadata records for containing this semantic data has closely paralleled the design of the bibliographic records used in libraries, since the two are effectively the same (with some minor differences due to

an electronic context). There are a number of existing metadata schemes (similar to database schemas, in that they define the structure of the metadata records) in use in the library and Internet worlds, most of which have been designed independently, and which have had varying degrees of success.

The majority of the metadata formats below specify a syntax in which records can be encoded in addition to the semantics expressed by the schema. Although this plays a part in the translation of records from one format to another, the semantic transformation is the harder to accomplish, and so the schema, not the syntax, is the more crucial. Another important distinction which should be made is between metadata formats and cataloguing rules. The latter, of which the Anglo-American Cataloguing Rules (Gorman and Winkler, 1988) are the best known example, specify the way in which data is entered into the bibliographic records specified by the metadata format. For example, the metadata format might require a field called 'author' which gives the creator of a document, but the cataloguing rules say how the name is to be written: *surname, first-name* or *first-name surname.*

A different strand of metadata classification has grown out of the work in the Artificial Intelligence field on knowledge representation. This type of metadata is typically a declarative language used with an ontology to express objects in a particular domain; the language (for example, KIF - see Section 2.3.1) roughly corresponds to the syntax used by a traditional metadata format, and the ontology roughly corresponds to the schema.

2.4.1 MARC

MARC (MARBI/ALA/LOC, 1996) stands for MAchine Readable Cataloguing, and is a standard format for representing bibliographic information (as might be found in a library catalogue) in an electronic form. Although designed for the library community, it has been adopted elsewhere as a convenient method for storing or exchanging data.

Originally designed by the Library of Congress, more than twenty other MARC formats have sprung up, including UKMARC (used by the British National Bibliography) and other national formats. As a reaction to the way in which the proliferation of national MARC formats have impeded the exchange of data, an international format, UNIMARC, was developed in the late 1970s.

MARC is an extremely rich metadata format, containing many hundreds of different fields. The data within these fields is highly structured, so that the components of an author's name may be extracted, for example. This complexity comes at a price; MARC records are time consuming to write and extremely difficult to generate automatically (the vast majority of library catalog records were originally written by humans).

2.4.2 IAFA

IAFA templates (Deutsch et al., 1995) were devised by the Internet Anonymous FTP Archive (IAFA) Working Group of the Internet Engineering Task Force (IETF) as a means for describing the contents of anonymous FTP archives (Beckett, 1995). The Internet Draft in which they were proposed has since expired, but has influenced the development of a number of other metadata schemes, such as SOIF and ROADS templates (more about which below).

An IAFA record is a field-based entity in which each field or *data element* is a discrete piece of information about some resource, along much the same lines as the fields in a MARC record. Unlike MARC, fields are not broken down into subfields when further describing an element. Also, there are certain classes of data elements, such as contact information, which always occur together. IAFA templates define the notion of a *cluster*[1], which allows these classes to be referred to in a shorthand manner.

2.4.3 Summary Object Interchange Format

SOIF (Hardy et al., 1996) is the native metadata format used by the Harvest system (see Section 2.2.2) to summarise the contents of the resources it holds. SOIF is a simple metadata scheme which extracts only a few properties from the resources (typically author, title, keywords, abstract and a description) and gives over most of its fields to information about the indexing process (the entity which gathered the properties, the time of gathering, the time until the record is to be discarded).

[1]Not to be confused with the conventional IR definition of this term as a group of related documents.

2.4.4 Platform for Internet Content Selection

Unlike the other Internet-based metadata schemes, PICS (Resnick and Miller, 1996) was not originally intended as a tool for resource discovery, but rather a tool for information filtering. PICS was conceived as a means by which parents could prevent their children from viewing certain types of information resources on the Internet, depending on some *rating* attached to the resources. The ratings are assigned by a number of *rating services* (Miller et al., 1996), individuals or organisations which provide *content labels* (Krauskopf et al., 1996) for resources on the Internet (eg. RSACi (Martin and Reagle, 1996)).

The labels provided are based on a *rating system* which specifies the dimensions used for labelling (eg. the attributes in the metadata record, in this case things like the severity of obscene language), the scale of allowable values for each dimension and some description of the criteria used in assigning values. In addition to the attributes specified by the rating service, a number of attributes are included in all PICS labels, and are automatically assigned (eg. the assigning service, the date of label creation).

2.4.5 Dublin Core

At present, the field of Internet metadata is undergoing its first round of standardisation, with the Dublin Core Metadata Set (DCMI, 1999) as the new standard scheme. Compared to expressive formats like MARC, DC is extremely simple, but this is in keeping with its positioning as a resource metadata scheme which is readily usable by the majority of authors on the World Wide Web. Its simplicity also lends it to being used as an interoperability format between the other schemes used to describe electronic resources.

Unlike the other metadata formats, the Dublin Core does not define a single encoding; the aim is to use native encodings to incorporate the metadata into a wide variety of resources. For example, DC data may be inserted into the `<META>` tag in an HTML (Raggett, 1997) document, encoded in a PICS label, written in a SGML (Goldfarb, 1990; ISO, 1986) DTD or expressed in RDF.

2.4.6 Resource Description Framework

RDF (Lassila and Swick, 1999) is a foundation for Web-based metadata which emphasises facilities for automated processing. Designed for generality, the W3C

intends to use RDF for a number of purposes, including resource discovery and cataloguing, content rating (see PICS in Section 2.4.4), knowledge sharing between software agents and digital signatures. RDF is divided into three parts:

Syntax: RDF uses a common encoding based on XML(Bray et al., 1998), an eXtensible Markup Language derived from SGML.

Model: The RDF model is the basic ontology used by all RDF records, and can be used to represent both traditional attribute value pairs, and relationships between resources.

Schema: RDF uses the RDF Schema language (Brickley and Guha, 2000) to define schemas (classes of resource and the properties which exist between resources) in the RDF model.

This separation of encoding from schema (ontology) simplifies many of the problems inherent in translating metadata records from one format to another.

RDF is an important component of the Semantic Web (Berners-Lee et al., 2001), the next stage in the development of the World Wide Web. A simple summary of the goals of the Semantic Web effort is that it aims to use the distribution mechanisms of the World Wide Web to build a large scale distributed knowledge base.

2.5 Hypertext and Hypermedia

The origins of the modern hypertext system are considered by many to lie with the Memex described by Bush (1945). Although revolutionary in its outlook, this system belies its pre-computer networking origins, for it effectively serves only a single user at one time and stores all of its data locally.

The first distributed hypertext appeared some twenty years later, albeit in embryonic form, with Ted Nelson's Xanadu system(Nelson, 1987), remembered chiefly for its first-class links (objects in their own right, stored separately from the documents they annotate) and the notion of *transclusion* (transparent quotation through inclusion, rather than through the copying of the quoted data).

An important notion in hypermedia systems is that of *open hypertext* or *open hypermedia*, the general term for systems which store links separately, and has lead to a number of further developments: first-class aggregate documents and links without fixed endpoint, such as *generic links* which construct a temporary endpoint on the fly which matches some phrase, or *functional links* which calculate their destination when invoked. A key publication which has affected much of the

development of open hypermedia systems is the Dexter model (Halasz and Schwartz, 1990), an early attempt at a comprehensive formal model and characterisation of hypertext and hypermedia systems.

The reified links which are a characteristic of open hypermedia systems are a very structured type of data, and their collection in *linkbases* makes them amenable to searching in a similar manner to metadata or directory records.

Distributed hypermedia has only properly come of age in the last fifteen years with the advent of affordable networking, and in that time there have been several systems which are important in their approaches to distribution.

2.5.1 Xanadu

Although Xanadu (Nelson, 1987) never progressed beyond a prototype, the published details of the system included several distribution-related features. Xanadu uses a single, extendible addressing scheme for all servers, users, document and even bytes, and has a published protocol (FEBE - the front-end/back-end protocol) for communications between the user's client program and the server. Unfortunately, many of the features most crucial to distribution remained unpublished for many years, including the BEBE (back-end/back-end) protocol used to forward client requests (when interviewed, Nelson (2001) confirmed that the BEBE protocol work had not been satisfactorily completed) and the Enfilade algorithm used to search for objects in the servers (an early version has since been published under the aegis of the open source Udanax[2] project).

2.5.2 Microcosm-TNG

Microcosm-TNG (Goose et al., 1997) was a further development of the Microcosm (Fountain et al., 1990) open hypermedia system which added distribution. The original Microcosm, which had been designed for a single user accessing a collection of multimedia data, was composed of a group of communicating processes or *filters* which transformed the stream of requests originating from the user's client application.

Microcosm-TNG used a more sophisticated communications model which allowed these processes to be based on different machines, and is designed for use at enterprise level. In addition, the processes of different users could communicate, enabling collaborative working. Amongst the processes were processes dedicated

[2] `http://www.udanax.org/`

to message routing, process management and brokerage, in a way which has anticipated the design of the largely CORBA-inspired agent environment used by FIPA (Microcosm-TNG's message router is equivalent to FIPA's ACC, while the brokerage component is equivalent to the FIPA DF – see Section 2.3.2).

2.5.3 HyperDisco

In its degree of distribution, HyperDisco (Wiil and Leggett, 1996) is an open hypermedia system similar in scope to Microcosm-TNG. HyperDisco is designed to work with a medley of different tools and information sources which are abstracted and integrated to create a uniform system. Although HyperDisco has a distributed *hyperbase management system*, there is no mention of the distributed search methods which are to be used to search this system; it is assumed that an exhaustive search will be used due to the expected size of a HyperDisco system.

2.5.4 Hyper-G/HyperWave

The Hyper-G (now HyperWave) system (Kappe, 1991) has been design for wide scale distribution from the outset, using an efficient and robust algorithm for circulating changes between constituent servers (i.e. maintaining consistency between servers). This algorithm, *p-flood* (Kappe, 1994), plays a similar role in Hyper-G to that of the Common Indexing Protocol in RWhois systems, that is as a method for facilitating query routing by passing indexing or other information between servers. p-flood is used to propagate changes to links and nodes through a Hyper-G system.

Information about surface links (links from one server's resources to another server's resources) are passed to all interested parties, with a probabilistic parameter governing the number of redundant copies which are transmitted in an attempt to forestall failures due to poor connectivity. Hyper-G uses this system to propagate linking information with the aim of providing the sort of referential integrity required by the Dexter model (Halasz and Schwartz, 1990).

2.5.5 World Wide Web

The World Wide Web (Berners-Lee et al., 1994a) needs little introduction, being the single largest hypertext system yet built. Unlike the other hypermedia systems in this section, the Web has a very simple linking model which is limited to static,

embedded, non-reified links. The Web's success as a distributed hypermedia system is largely due to the simplicity and robustness of its Uniform Resource Locator addressing scheme (Berners-Lee et al., 1994b) which gains its distributed characteristics from the Domain Name Service (Mockapetris, 1987a), which is used to resolve part of the URL.

The Web protocols (chiefly HTTP (Fielding et al., 1999)) do not contain any support for maintaining link integrity, unlike several other hypermedia systems. There have been proposals for systems which would provide such a capability, such as the ATLAS system described by (Pitkow and Jones, 1996).

Recent developments from the Web standards body, the World Wide Web Consortium (W3C) [3] have included XML (Bray et al., 1998), a subset of SGML, whose related specifications include XLink (W3C, 1999a) and XPtr (W3C, 1999b). These draft standards describe a more complex linking model for the Web which could support first-order links. Resolving those links may require a distributed search systems, but as yet no such system has been proposed.

As mentioned in Section 2.4.6, the W3C vision for the future development of the World Wide Web is that of a semantic web containing information imbued with machine-readable meaning (Berners-Lee et al., 2001), which goes far beyond the existing typeless associative hyperlinks of the Web.

2.5.6 DLS

The Distributed Link Service (Carr et al., 1995) uses Microcosm's linking model with the Web by interposing a proxy between the user's client software and the sever. This server intercepts requests, consults a linkbase and then rewrites the received page to reflect the results from the linkbase. Originally designed so that the linkbase selection were made from an explicit list, more recent developments (DeRoure et al., 1999) have used query routing to resolve links in an unordered collection of linkbases.

Related work to the DLS includes the COHSE project (Goble and Carr, 1999), a *conceptual open hypermedia system* which employs a knowledge base in the construction and selection of hyperlinks for the contextual annotation of documents.

[3] `http://www.w3.org`

2.5.7 Open Hypermedia Protocol

The Open Hypermedia Protocol (Reich et al., 2000) can be seen as the continuation of the work which begun with the Dexter model(Halasz and Schwartz, 1990), designing a rich data model and protocol to allow different hypermedia systems to interact. The distribution of OHP is on a par with that of Microcosm; it is possible to do, but there is no explicit support for the sort of distributed search required to make link resolution or resource discovery work.

2.5.8 Agents and Hypermedia

Agents have been used as a framework for distribution in several hypertext and hypermedia systems. The University of Michigan Digital Library Project (UMDL) (Birmingham, 1995), the Zuno Digital Library (Ferguson and Wooldridge, 1997) and the MNA project use agents as mediators which match user requests to appropriate collections of data, while the Voyager project (Dale, 1997; Dale and DeRoure, 1996) uses agency as a more general technique for distributing all aspects of an open hypermedia system. Perhaps the most ambitious agent-based hypermedia system is NIKOS (Salampasis, 1998; Salampasis et al., 1996), which has specialised agents for handling each of the base components (nodes, links, composites) in the storage layer of the Dexter model (Halasz and Schwartz, 1990).

2.6 Summary

In this chapter we have summarised prior work in the literature which is relevant to our study of query routing systems, or which informs the design of ontologies for our chosen application domains. In the following chapter we describe a model of a query routing system, and in Chapter 4 we investigate the effects that different network topologies (drawn from the exemplar distributed search systems in this chapter) affect the behaviour of a query routing system.

Chapter 3

A Model of Query Routing Search

3.1 Introduction

In this chapter, we describe a model of a query routing system which is based on functional aspects of query routing (effectiveness at answering queries) rather than on the performance aspects (efficiency of answering queries).

The distributed search systems considered in the previous chapter have the same common aim: to provide a means by which objects may be located in a distributed system given only a description of the properties which the objects should possess (bibliographic metadata, for example). This task is commonly known as *resource discovery*. The services which carry out the resource discovery task are variously considered as name resolution services, information retrieval services or brokerage services, but in each case the action of the system is the same, even though the acceptable parameters for its operation may differ[1].

The resource discovery task is a surprisingly common one, appearing in disciplines from information retrieval and library science to open hypermedia and the Web or distributed artificial intelligence. In all of these areas, a robust and reliable resource discovery service is a key requirement in managing system distribution, but the majority of such services are themselves not distributed. This presents a potential limit to the scalability of these distributed systems as it will become progressively harder to locate objects as the systems grow.

Many of the distributed systems for resource discovery considered in the previous chapter try to reduce the complexity of the task by discarding large numbers of candidate solutions in order to prune the search space. The exact method by

[1]Locating objects which only partially meet the criteria is more acceptable in information retrieval than in name resolution, for example.

which this is carried out varies, but the essence of the technique is to partition the search space (a necessary prerequisite for distribution anyway) and then to use the knowledge of the contents of these partitions to guide the processing of the query by selecting partitions which are particularly likely to contain satisfactory answers. As the system as a whole grows, the number of partitions should also increase, so that the size of the partitions remains more-or-less constant.

An ideal system for distributed resource discovery would process queries with maximum efficiency by pruning the search space to only those partitions which contained satisfactory solutions (and similarly attain maximum effectiveness by selecting all the partitions which contains solutions).

3.2 Problem Statement

In order to design an efficient and effective system for distributed resource discovery, we must first understand the variables which affect the performance of such a system. We make the following assumptions about the composition of the distributed system.

- The system is composed of a number of entities capable of performing computations, which we shall call *servers.*
- The data objects which are to be the target of the resource discovery task are distributed amongst the servers in the system. We call a server to which have been allocated data objects a *data server.*
- The data objects allocated to a given data server have some similarities and are said to form a *cluster* (i.e. a collection of related documents).
- It is possible to construct an expression which represents the contents of a cluster, which we call a *cluster representative.* The centroids used by the WHOIS++ system described in Section 2.2.5 are one possible type of cluster representative.
- If a server contains the cluster representative for another server, it is said to have *forward knowledge* about that server.
- We call a server which has forward knowledge an *index server.*
- Two or more cluster representatives held by an index server may be combined to give an expression which represents the data objects which the index server has forward knowledge about.

	Relevant	Non-relevant	
Retrieved	$D_{rel} \cap D_{ret}$	$\neg D_{rel} \cap D_{ret}$	D_{ret}
Not retrieved	$D_{rel} \cap \neg D_{ret}$	$\neg D_{rel} \cap \neg D_{ret}$	$\neg D_{ret}$
	D_{rel}	$\neg D_{rel}$	D

Table 3.1: D is the set of documents in the system, D_{rel} the set of documents which are relevant to the query and D_{ret} the set of documents which are retrieved as a result of processing the query.

- An index server may have forward knowledge about the forward knowledge of another index server (a recursive definition to allow an arbitrary number of levels)

3.3 Foundations

If the aim of information retrieval or resource discovery systems is to find documents or objects which are relevant to a user's query, then *relevance* is perhaps the most important notion in information retrieval. Relevance has consequently been the subject of much research, but despite this attention, relevance remains a concept which is notoriously difficult to define to the satisfaction of all, a fact borne out by the large number of different definitions in the literature Mizzaro (1998). The ability of an IR system to retrieve relevant documents while keeping the number of irrelevant documents to a minimum is commonly known as the *effectiveness* of the system. As with relevance, there are a number of measures of effectiveness, of which we will be considering the most widely used, *precision* and *recall.* Finally, if the effectiveness of an IR system measures its ability to retrieve relevant documents, its ability to achieve this end with the minimum expenditure of effort is measured by its *efficiency.*

3.3.1 Retrieval Effectiveness

The definitions of precision and recall are based on a contingency table (Table 3.1) that partitions the set of documents in the system based on retrieval and relevance.

The *precision* (P) of a system is the number of records relevant to the query which were retrieved, expressed as a proportion of the total number of records retrieved. A precision of 1 means that all the records retrieved were relevant, a precision of 0 means that none were.

$$\mathrm{P} = \frac{|D_{rel} \cap D_{ret}|}{|D_{ret}|} \tag{3.1}$$

The *recall* (R) of a system is the number of records relevant to the query which were retrieved, expressed as a proportion of the total number of relevant records in the system. A recall of 1 means that all of the relevant records were retrieved, a recall of 0 means that none were.

$$\mathrm{R} = \frac{|D_{rel} \cap D_{ret}|}{|D_{rel}|} \tag{3.2}$$

Precision and recall are in an inverse relationship. If the set of records retrieved in response to a query is enlarged (by relaxing the criteria which are used to judge whether or not a record is retrieved, for example), there will be an increase in recall (because potentially more of the relevant records in the system are retrieved) at the expense of a decrease in precision (because there is potentially a greater proportion of irrelevant records in the retrieved set), and vice versa when the retrieval set is reduced in size.

3.3.2 Forward Knowledge Effectiveness

The forward knowledge which a server holds about another server can be viewed as a surrogate for the knowledge contained in the other server. We assume that the records in a server are closely related to each other and form a cluster. The records in a server are therefore likely to be relevant to the same requests, following the *cluster hypothesis* (summarised by van Rijsbergen (1979)).

The construction of such clusters is considered to be beyond the scope of this work; Jain et al. (1999) give a comprehensive review of clustering techniques. Of particular interest is the notion of *data abstraction*, by which a cluster is represented or described in a compact form. The construction of centroid-based cluster representatives for forward knowledge (as in WHOIS++ – see Section 2.2.5) is such an example of data abstraction.

As an example of a potential approach to clustering structured objects such as bibliographic records, Murty and Jain (1995) describe a scheme for the construction of clusters in the context of collection management. In this, objects in the collection (journal articles, for example) are represented by conjunction of weighted disjunctions, where the elements in each disjunction are node labels taken from a subject

	In cluster	Not in cluster	
In representation	$D_{clu} \cap D_{rep}$	$\neg D_{clu} \cap D_{rep}$	D_{rep}
Not in represention	$D_{clu} \cap \neg D_{rep}$	$\neg D_{clu} \cap \neg D_{rep}$	$\neg D_{rep}$
	D_{clu}	$\neg D_{clu}$	D

Table 3.2: D is the set of documents in the system, D_{clu} the set of documents which lie within a cluster and D_{rep} the set of documents which are represented by the cluster representative.

classification hierarchy (eg. Dewey Decimal or the ACM Computing Reviews classification). The representation of a single object is a summary of the subjects which it deals with (for example, this thesis could be described as being 40% *I.2.11 - Distributed Artificial Intelligence*, 20% *I.2.4 - Knowledge Representation Formalisms* and 40% *H.3.3 - Information Search and Retrieval* or *H.3.7 - Digital Libraries*). In this approach, the descriptions are clustered by using a complete-link hierarchical clustering algorithm with a similarity measure which takes into account the structure in the classification scheme.

An instance of forward knowledge contains two things; a reference to a server, and an expression which represents the contents of that server. The ability of forward knowledge to accurately direct search depends on how closely the expression reflects the underlying cluster. In our previous work (Gibbins, 1997), we defined the measures of *completeness* and *faithfulness* for describing the effectiveness of forward knowledge. These measures are similar to those used for retrieval effectiveness, and are constructed by partitioning the set of records into those which fall within the cluster, and those represented by the expression (see Table 3.2).

The *completeness* ($\mathcal{C}$) of a forward knowledge representation is a measure of how much of a cluster's contents it describes. The most general representation which claims to describe the entire contents of a cluster is trivially complete.

$$\mathcal{C} = \frac{|D_{clu} \cap D_{rep}|}{|D_{clu}|} \tag{3.3}$$

The *faithfulness* ($\mathcal{F}$) of a forward knowledge representation is a measure of how many of the documents described by the representation are not in the clusters it describes. The most general representation is complete, but is not faithful because it falsely claims that the cluster contains a number of records which in fact it does not.

$$\mathcal{F} = \frac{|D_{clu} \cap D_{rep}|}{|D_{rep}|} \tag{3.4}$$

There is a tradeoff between completeness and faithfulness which is analogous to that between precision and recall (described in Section 3.3.1), in that rewriting the cluster representative in order to relax the criteria for membership in a cluster is likely to correctly categorise more of the records which belong in the cluster (increasing completeness) at the expense of incorrectly more records which do not belong in the cluster (decreasing faithfulness).

3.3.3 Relevance and Structural Matching

The definitions of relevance in the literature (summarised in Mizzaro (1998)) are largely subjective in nature, being concerned with the user's self-assessment of their information need and the degree to which the retrieved documents satisfy this need. In particular, users possessed of different information needs may still formulate the same queries and so will rate the relevance of documents differently.

These subjective definitions of relevance present some problems in assessing the effectiveness of distributed search systems. Subjective relevance is of most use in systems which offer full text searching on largely unstructured documents, but many of the distributed search systems contain only highly structured resources such as white pages (directory) entries or hypertext links. With these sorts of resources, it is possible to give a purely objective definition of relevance based on structural matching such as unification or subsumption.

Unification is a general-purpose pattern matching algorithm which is widely used in traditional AI applications and languages such as Prolog. Unification compares the structures of two expressions containing variables, and generates a substitution which binds those variables to values which will make one expression equivalent to the other. A one-way matching would take a pattern and a target and find a substitution such that:

$$pattern \bullet substitution \equiv target$$

For example, a substitution which binds $X \mapsto 0$ and $Y \mapsto s(0)$ will unify the expressions $s(X) + Y$ and $s(0) + s(0)$:

$$[s(X) + Y] \bullet \{0/X, s(0)/Y\} \equiv [s(0) + s(0)]$$

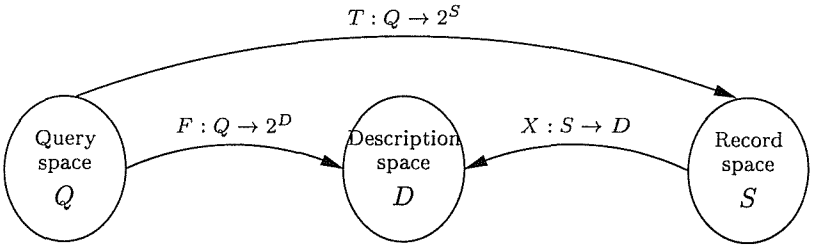


Figure 3.1: Retrieval Mapping with Indexing Function

For the purposes of examining search in this chapter, we are not interested in the values taken by the variables, and for clarity will write $p \sqcap t$ to denote that p unifies with t.

Subsumption is a form of structural matching which is not symmetric (unlike unification, where $s \sqcap t \iff t \sqcap s$). An expression subsumes another (denoted $s \sqsupseteq t$) if it is more general. For example, the expression $s(0) + s(0)$ is subsumed by the expression $s(0) + s(X)$ (and not vice versa, because the latter is more general), while the expressions $s(0)+X$ and $s(Y)+s(0)$ do not subsume each other (although they do unify with each other).

3.3.4 Modelling Retrieval

The set theoretical view of information retrieval has been studied in the past because it provides a simple formalism of the IR process. Salton (1979) and others introduced a model of the mapping from the set of queries Q to the set of documents S (the record space), as shown in Figure 3.1. This incorporated an indexing function $X : S \to D$ which assigns descriptions to documents. The overall retrieval process is represented by the function $T : Q \to 2^S$ which maps queries to sets of relevant documents, and can be composed from the function mapping queries onto the descriptions of documents which might be relevant, $F : Q \to 2^D$, and the inverse of X.

We refine this model by assuming that document descriptions and queries (also cluster representatives) are all described using the same language, so retrieval may be modelled as a mapping from the expressions of that language to the set of documents (see Figure 3.2). We take the set Q to be the set of possible expressions in that language, both those which denote a single document, like metadata records, and those which denote many documents, like queries or cluster representatives. The retrieval function T maps these expressions onto the records which they denote. The indexing function X maps S, the set of records, onto D, now a subset of Q,

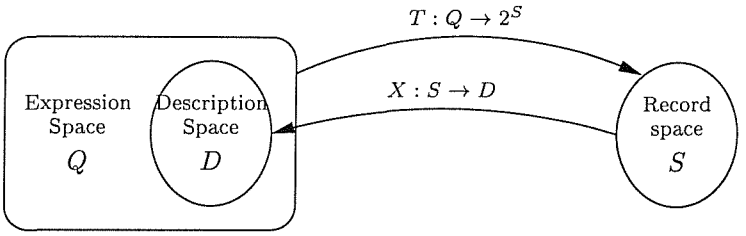


Figure 3.2: Refined Retrieval Model

such that for each element in S there is one and only one expression in this subset that describes it, but also that a description may be shared by several records.

The function X defines an equivalence relation on S such that documents $s_i, s_j \in S$ are equivalent if and only if $X(s_i) = X(s_j)$. A justification for this can be taken from the library world; two books may be physically different entities but share the same bibliographic data in a library catalogue since they are both the same edition of the same work and so equivalent. In this respect, the expressions in D are similar to the descriptive names introduced by Neufeld (1989) (although the latter denote objects unambiguously, while the former denote a class of equivalent objects).

The function T is such that, if its domain is restricted to D, it is equivalent to the inverse of X. This allows us to define the *unifies* ($\sqsupset : Q \times Q$) and *subsumed by* ($\sqsubseteq : Q \times Q$) relations as follows:

$$x \sqsubseteq y \iff T(x) \subseteq T(y) \tag{3.5}$$

$$x \sqsupset y \iff T(x) \cap T(y) \neq \emptyset \tag{3.6}$$

The set Q has a partial order under $\sqsubseteq$ and forms a lattice bounded by $\top$ (top, the expression which denotes all records) and $\bot$ (bottom, the expression which denotes no record), being isomorphic with 2^S under $\subseteq$ bounded by S and $\emptyset$. It is useful to define the *most general unification* ($\sqcap : Q \times Q \to Q$) and *most specific generalisation* ($\sqcup : Q \times Q \to Q$) operations (the meet and join of this lattice) as:

$$T(x \sqcap y) = T(x) \cap T(y) \tag{3.7}$$

$$T(x \sqcup y) = T(x) \cup T(y) \tag{3.8}$$

In stating that both queries and forward knowledge expressions are drawn from

Syntax	Semantics	Description
A	$A^{\mathcal{I}} \subseteq \Delta$	primitive concept
R	$R^{\mathcal{I}} \subseteq \Delta \times \Delta$	primitive role
$\top$	Δ	top
$\bot$	$\emptyset$	bottom
$\neg C$	$\Delta \setminus C^{\mathcal{I}}$	complement
$C \sqcap D$	$C^{\mathcal{I}} \cap D^{\mathcal{I}}$	conjunction
$C \sqcup D$	$C^{\mathcal{I}} \cup D^{\mathcal{I}}$	disjunction
$\forall R.C$	$\{x : \forall y.R^{\mathcal{I}}(x,y) \implies C^{\mathcal{I}}(y)\}$	universal quantification
$\exists R$	$\{x : \exists y.R^{\mathcal{I}}(x,y)\}$	existential quantification

Table 3.3: Syntax and semantics of $\mathcal{ALU}$

Q, we make the assumption here that it is possible to formulate a query which completely specifies a single record; if this is not the case, queries should be drawn from $Q \setminus D$. The function T defines which documents are relevant to a given query, and unification and subsumption are used to determine if a forward knowledge expression is relevant to a query. Existing distributed search systems use both unification and subsumption to determine whether a record or template is relevant to a query – the effect of the choice of relation on the behaviour will be dealt with later.

This model of relevance resembles a version of the simple description logic $\mathcal{ALU}$(Donini et al., 1996). Description logics (DLs) arose out of the development of frame-based knowledge representation languages, for which they provide a formal foundation. DLs consist of *concepts*, *roles* and *individuals*; concepts are expressions which describe a set of individuals which have some commonality. Individuals are related to each other by roles, which describe the properties of individuals, and which are also used in the formation of concepts.

Traditionally, the semantics of a description logic are described through the use of an interpretation consisting of a domain Δ and an interpretation function $\cdot^{\mathcal{I}}$ which maps individuals onto elements of Δ, concepts onto subsets of Δ and roles onto subsets of $\Delta \times \Delta$. The concepts in $\mathcal{ALU}$ (denoted by C and D) are constructed according to the syntax rules in Table 3.3 and have the given extensions under $\cdot^{\mathcal{I}}$. Our simple model of information retrieval is equivalent to a subset of $\mathcal{ALU}$ which does not contain roles; S is our domain, Q is the set of concepts and T is an interpretation function. Following the standard description logic conventions, we will subsequently refer to the set of records in our model as Δ, the set of queries and descriptions as C (the set of all concepts) and the retrieval function as $\cdot^{\mathcal{I}}$.

We can express the measures for forward knowledge effectiveness using this model. Here we measure the effectiveness of an expression $t \in Q$ at representing a clustered set of documents, D_{clu}.

$$\mathcal{C}(t) = \frac{|D_{clu} \cap t^{\mathcal{I}}|}{|D_{clu}|} \tag{3.9}$$

$$\mathcal{F}(t) = \frac{|D_{clu} \cap t^{\mathcal{I}}|}{|t^{\mathcal{I}}|} \tag{3.10}$$

Defining relevance in this way as a yes/no proposition is, however, at odds with many of the existing notions about relevance; two expressions either unify or do not, but there are many shades of relevance between a completely relevant document and a completely irrelevant one. The choice of such binary relevance also affects the values taken by the effectiveness measures; if unification is also used to determine which records are to be retrieved, all searches are trivially precise (= 1) because it is possible to retrieve only completely relevant documents. We can expand our definition of relevance to include non-binary values by drawing on the concepts introduced by fuzzy set theory.

Fuzzy set theory (Klir and Yuan, 1995) is a generalisation of classical (crisp) set theory in which the degree of membership of an element in a set can range from entirely to not at all. Formally, a fuzzy subset A of a crisp set X is defined as a set of tuples $\langle x, \mu_A(x) \rangle$ where $x \in X$ and μ_A is a membership function $\mu_A : A \rightarrow [0, 1]$ which indicates the degree of membership of x in A. When the values taken by μ_A are drawn from the set $\{0, 1\}$, the behaviour of A under the familiar set operations is the same as if it were a crisp subset. In the interval between, the set behaviour is defined in terms of the membership function. An *alpha-cut* A_α of a fuzzy set A is a crisp set which contains those elements which have a degree of membership in A of at least α.

$\mu_{A \cup B}(x)$	$=$	$\max(\mu_A(x), \mu_B(x))$	union
$\mu_{A \cap B}(x)$	$=$	$\min(\mu_A(x), \mu_B(x))$	intersection
$\|A\|$	$=$	$\sum_{x \in X} \mu_A(x)$	cardinality
$A \subseteq B$	$\iff$	$\forall x \in X, \mu_A(x) \leq \mu_B(x)$	subset
A_α	$=$	$\{x : \mu_A(x) \geq \alpha\}$	alpha-cut
$A_{\alpha+}$	$=$	$\{x : \mu_A(x) > \alpha\}$	strict alpha-cut

We can extend our model to include fuzzy characteristics following the work of Straccia (1998) on fuzzy description logics. We redefine the range of the interpretation function $\cdot^{\mathcal{I}}$ to be the set of membership functions $\Delta \rightarrow [0,1]$ so that we can now talk about documents in Δ being 'mostly relevant' or 'slightly relevant' to expressions in C. The definition of subsumption (3.5) remains as before, but unification is now defined as in Equation 3.11. The intuition behind this definition is that two concepts should be considered to be capable of being unified if there is any overlap between the (fuzzy) extensions of those concepts; the alpha-cut at zero selects all elements of the domain which have at least some degree of membership in the intersection of the concept extensions.

$$x \sqcap y \iff (x^{\mathcal{I}} \cap y^{\mathcal{I}})_{0+} \neq \emptyset \tag{3.11}$$

Having defined the notion of relevance using a fuzzy interpretation function, we can now express our relevance measures in terms of this function. We can give the user a means of trading precision and recall off against each other by defining the set of retrieved documents to be the alpha-cut of the set of relevant documents. There can be no fuzziness in the set of retrieved documents; documents are either retrieved or not. The set of relevant documents, however, need not be a crisp set, because the documents may be of varying degrees of relevance to the user's information need (the membership degree of an element in the set indicating relevance). We use an alpha-cut to represent the retrieved document set because it is a crisp set based on the relevant set; the value of α is a parameter which can be tuned in order to trade off the precision of a query against its recall by retrieving more or fewer records. The measures of precision and recall can therefore be defined as follows:

$$\mathrm{P}(q) = \frac{|q^{\mathcal{I}} \cap q^{\mathcal{I}}{}_{\alpha}|}{|q^{\mathcal{I}}{}_{\alpha}|} \tag{3.12}$$

$$\mathrm{R}(q) = \frac{|q^{\mathcal{I}} \cap q^{\mathcal{I}}{}_{\alpha}|}{|q^{\mathcal{I}}|} \tag{3.13}$$

However, our chosen application domains typically use structured records for which matching techniques such as subsumption or unification are appropriate. The nature of relevance in information retrieval systems is domain dependant; the fuzzy model given above is appropriate for those domains which require a sliding scale of relevance, but for our chosen domains its crisp counterpart based on conventional description logics will suffice.

3.3.5 Retrieval Efficiency

Information retrieval research has traditionally focussed on effectiveness as a measure for the assessment of information retrieval systems, so concentrating on improving the accuracy of the results that such a system returns. Indeed, efficiency is commonly considered to be a lesser concern to effectiveness because a system which returns poor quality results is of little use, regardless of how quickly it returns those results. A recent study by Frieder et al. (1999) notes that one recent collection of seminal research papers (Sparck Jones and Willett, 1997) did not contain a single paper on efficiency considerations.

The advent and rise of distributed information retrieval has changed this state of affairs to some extent. Web search engines are under considerable pressure to deliver their results quickly; the process of evaluating a query is perceived by many users as being no more complex than that of following a link, so the search engine should return its results within an order of magnitude of the time taken to traverse a link. Similarly, the impact that the information retrieval system has on its environment (its use of network resources while building its index and processing queries, for example) is vitally important. A web search engine which places a heavy load on the network would be deemed unacceptable and antisocial.

Existing studies of information retrieval efficiency have made use of measurements such as the time taken to process queries or the size of an index in bytes. While wholly appropriate for an empirical study of an existing system, they are less so when modelling the behaviour of a hypothetical system. We choose a set of more abstract complexity measures of communication and space. While these are strongly related to the real-world measurements, they are more amenable to a study which does not consider the added modelling complexities of network latency and the like.

Query Complexity

When a query is submitted to the system, the number of request and response messages which are generated between system components (the message traffic) is a measure of the communication complexity of query processing. The rate at which this grows relative to the number of servers in the network gives an indication of its scalability. In addition, the individual message traffic for each server in the system can be used to show the presence of bottlenecks, servers which contribute disproportionately to the global message traffic. We do not explicitly specify whether the

system operates by means of referrals or by delegation since a single referral generates as many messages as a single delegation. However, a system which operates by referrals is less prone to making redundant queries (where a server is queried more than once) because the state of the query is stored in one location (the client), making it easy to remove duplicates. A system which uses delegation could be expected to send more redundant messages, but the exact effect cannot be quantified without more detailed knowledge of the exact structure of the forward knowledge graph and the query being asked.

We consider our hypothetical systems to operate in synchronous parallel rounds, one round being the time for each entity in the network to communicate with one or more of its neighbours. The running time (time complexity) for query processing is the number of rounds required until the client receives an answer to its query.

Control Complexity

Although the message traffic generated by submitting queries can be used to measure system load under normal operations, it would not be possible to process queries at all if there were no forward knowledge in the system. This forward knowledge must be built up by passing messages which contain forward knowledge summaries between servers.

The forward knowledge graph affects the query message traffic by constraining the types of referral which may be issued. However, the forward knowledge graph must be constructed by an initial exchange of messages. These messages are equivalent to the control messages which allow a communications network to build its routing tables; the number of control messages sent by the system gives the communication complexity of the forward knowledge building operation. We consider only those control messages sent when a system is started; control messages sent as the result of dynamic changes over time to the resources held by the servers are not considered.

As with query message traffic above, we measure time complexity by counting the number of rounds taken for the system to complete its task. In this case, the system must *converge*, reach a state where each server has sufficient forward knowledge that all resources that are relevant to a query are reachable).

Update Complexity

While the control complexity of a system measures the effort required to build the forward knowledge network from scratch, frequently we will be presented with a situation in which a single server has changed its holdings and wishes to advertise this fact. In this situation, we need to know the complexity of propagating such an incremental update, and in particular how it compares to the complexity of building the forward knowledge network from fresh.

Again, time complexity is measured by counting the number of rounds until convergence and communication complexity by counting the number of messages sent within the system in the course of propagating the update.

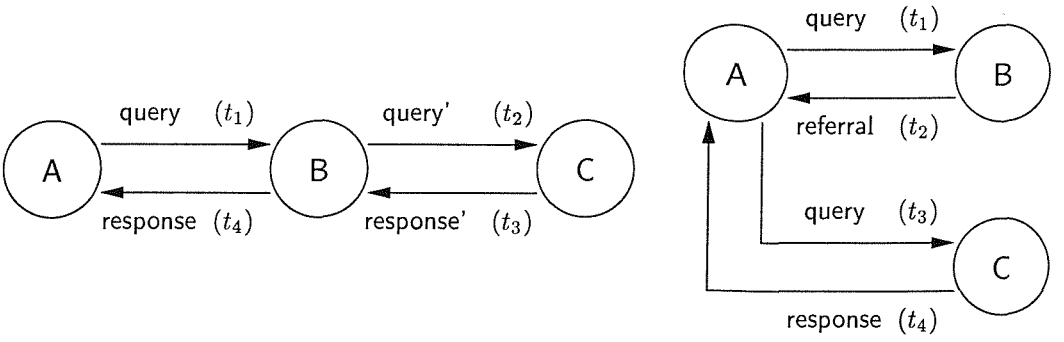
Routing Table Size

If the number of control messages shows the communications overhead inherent in building the forward knowledge graph, the size of the routing tables held by each server as a result of the control messages above shows the space required to store the routing information. A system's control message traffic could be largely redundant, so a server would not retain each message it received; this measure gives the size of the 'useful' control information.

3.4 Delegation and Referral

Although forward knowledge controls the distribution scope of the processing of a query, it does not control how the query is distributed. Delegation and referral are two commonly used techniques by which the processing of a query may be distributed through a system. When a query is presented to a server which is aware (by virtue of its forward knowledge) that there is another server better suited to dealing with the query, it can choose to propagate the query by either method. On delegating a query (Figure 3.3(a)), the server asks the query of the second server, and on receiving an answer, passes this back to the client which issued the request, so delegating responsibility for processing the query to the second server. Alternatively, the server might instead refer the client to the second server (Figure 3.3(b)), informing it of its existence and its suitability to the task in hand, and leaving the client to carry out further work.

Delegation and referral are known by different names in the agent world, namely brokerage and matchmaking. Agents which provide either facility are collectively



(a) Delegation **(b)** Referral

Figure 3.3: Delegation and Referral

known as *middle agents* or *mediators*, because they are interposed between agents which provide information and those agents which seek it. Brokering in particular may go beyond information retrieval concerns; a broker may be presented with a task which it then recruits a team of agents to fulfill, using a plan of its own formulation. The treatment of middle agents in Decker et al. (1996) identifies a difference in the ways that a client interacts with matchmakers and brokers, based on the semantics of the messages exchanged.

When talking to a matchmaker, a client sends a message asking "who is able to service my request". By comparison, when a client speaks to a broker, it sends a message saying "service this request". The content of these messages is observably different, even if both messages are requests to be sent information.

This is at odds with the behaviour of a number of existing search systems (most notably DNS) in which a service request may be responded to with either referrals or brokered answers. In effect, the queries sent by DNS clients are hybrid entities which informally might correspond to the question "tell me what you know about x and/or tell me who knows about x".

The example given by Decker et al. (1996) also uses different KQML message types for communicating with matchmakers and brokers (`ask-all` and `stream-all` respectively), but this is not relevant to the above argument; the message types have the same semantics (the effects on the listener, as noted informally by ?), and differ only in the mode of delivery of their responses (`ask-all` requires an agent to send all its responses in one message, whereas `stream-all` returns a stream identifier which the recipient may use to control the rate at which it receives the responses).

Delegation and referral are different coordination strategies to the distribution of queries, but can be driven from the same forward knowledge. The difference

between the strategies lies in which entity has control over the progress of the query. A server which delegates a query controls which data is returned to the client, and so may aggregate solutions from other sources before dispatching them to the client. The client may however be possessed of knowledge which might aid the satisfaction of the query (for example, which servers are considered to be sources of trustworthy information) and which cannot be used because the server has sole control over which sources are used. Conversely, while issuing referrals grants clients the power to decide how and where a query is processed, they are under no obligation to follow the suggestions made by the issuing server. In Decker et al. (1997), the authors go further in their characterisation of middle agents by identifying and classifying a variety of middle agent roles based on which agents initially know the preferences of the requesting agents and the capabilities of the providing agents. This privacy based model of the connection problem (finding other agents with the capabilities you need) investigates the distribution of initial knowledge in the system, which in turn gives agents the necessary means to control the direction and scope of the search.

Delegation and referral represent the extremes of a continuum of distributed search techniques. At one end, the servers maintain the state of processing of the query (the list of servers which have been queried so far, and those which have been identified as possibly relevant, but which have yet to be queried), while at the other this is controlled by the client. Each has advantages over the other in certain circumstances, as will be further explained in Section 4.5, so there is some reason to believe that a search system with mutual state – shared between servers and clients – could combine the best features of both.

3.5 Modelling Forward Knowledge

The organisation of a distributed search system is the key factor which determines its effectiveness and efficiency. If a given query is to be processed with perfect ($= 1$) recall, it must be presented to all of the nodes which contain relevant data. On the other hand, we wish to use as few system resources as possible in the processing of the query, so the query should be presented to as few nodes as possible. Thirdly, we do not want the burden of query processing to rest unevenly on particular nodes; as far as is possible, all of the nodes in the system should play an equal role in the processing of queries. We have used a graph-based approach to model the network

of forward knowledge in a query system, but see (van Eijk et al., 2000) for work which uses the Kripke structures of a modal logic to model network topologies.

3.5.1 The Forward Knowledge Graph

If a query is to be presented to a server which has relevant data, there must exist a chain of referrals from the initial server (where the query was first asked) to the target server. For these referrals to be generated, the templates for the forward knowledge from which they are generated must be relevant to the query. As a precursor to determining the effects of forward knowledge distribution on the performance of distributed search systems, we can model the forward knowledge and identify useful properties by considering the graph made by the forward knowledge (this work is derived from the path calculus described by Cormen et al. (1990, p.570)).

The forward knowledge graph is a multigraph $G = \langle V, E \rangle$ with directed edges which are labelled with (possibly non-unique) expressions from C (the forward knowledge templates) and nodes which are also labelled with expressions from C (each data server holds some subset of Δ, so the node labelling expressions from C are cluster representatives for each server in the system). We represent the edge labelling with the functions $\lambda : V \times V \to C$ and the node labelling with the function $\nu : V \to C$. Because λ is defined over the domain $V \times V$, we take $\lambda(u, v) = \bot$ if $\langle u, v \rangle \notin E$ (i.e. non-existent edges are treated as though they existed and had been labelled with *bottom*, the expression which does not subsume any expressions).

The notion of edge labels may be extended to label paths with the most general expression which will satisfy all of the edges in the path; if this expression is relevant to a query, sufficient referrals will be generated to allow the propagation of the query to the path end (which presumably holds records which are relevant to the query). If we have a path p from v_0 to v_n (written as $v_0 \rightsquigarrow^p v_n$ and shown in Figure 3.4), the $\sqcap$ operator (3.8) can be used as an extension operator to give a path label determined by:

$$\lambda(p) = \lambda(v_0, v_1) \sqcap \lambda(v_1, v_2) \sqcap \cdots \sqcap \lambda(v_{n-1}, v_n) \tag{3.14}$$

Although the edges would be traversed in order from $\langle v_0, v_1 \rangle$ to $\langle v_{n-1}, v_n \rangle$ when a query is processed in a real system, $\sqcap$ is associative on C. Since $\top$ is an identity for $\sqcap$, the system $\langle C, \sqcap, \top \rangle$ is monoid.

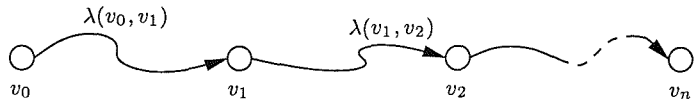


Figure 3.4: Path extension

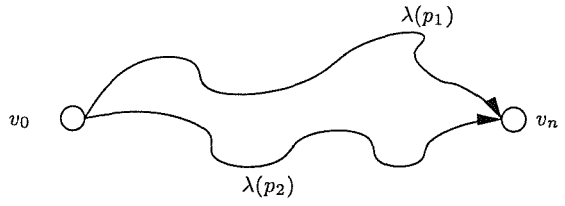


Figure 3.5: Summary labels for parallel paths

The forward knowledge graph may contain more than one path from the initial node to the target node, so we must also define a summary operator which can combine the labels of several paths in order to give an expression which denotes the queries which can be answered by the target node, starting at the initial node. For systems which use unification ($\sqcap$) for their relevance relation, we can use the most specific generalisation ($\sqcup$) as our summary operator. The aggregate label $\lambda(v_0 \rightsquigarrow^* v_n)$ for all of the paths $v_0 \rightsquigarrow v_n$ is given by:

$$\lambda(v_0 \overset{*}{\rightsquigarrow} v_n) = \bigsqcup_{v_0 \rightsquigarrow^p v_n} \lambda(p) \tag{3.15}$$

However, $\sqcup$ is not immediately appropriate as a summary operator for those systems which use subsumption ($\sqsubseteq$) as their relevance relation; in this group of systems are included Nomenclator (Ordille, 1998) and RWhois (Blacka et al., 1998)), which both require that only templates which completely cover a query are used for generating referrals. If we envisage a graph with two paths $v_1 \rightsquigarrow^{p_1} v_n$ and $v_1 \rightsquigarrow^{p_2} v_n$, labelled with λ (Figure 3.5), the summary label of the two paths is $\lambda(p_1) \sqcup \lambda(p_2)$. If we formulate a query $q \in C$ such that $q \sqsubseteq \lambda(p_1) \sqcup \lambda(p_2)$, $q \not\sqsubseteq \lambda(p_1)$ and $q \not\sqsubseteq \lambda(p_2)$, q cannot be satisfied by v_n from v_0 because there is no single path that can be traversed, despite what the summary label suggests.

If we allow queries to be decomposed into smaller queries, $\sqcup$ does hold as a summary operator. If we formulate $q \in C$ and then break it down into $q_1, q_2 \in C$ such that $q = q_1 \sqcup q_2$, $q_1 \sqsubseteq \lambda(p_1)$ and $q_2 \sqsubseteq \lambda(p_2)$, each of the two parts of the query can be satisfied by traversing a different path to v_n from v_0.

In addition, if we use $\sqcup$ as a summary operator, the issue of contradictory summaries, where the generated summary has an empty extension, does not arise. The expressions which are summarised each denote a class of entities, and the

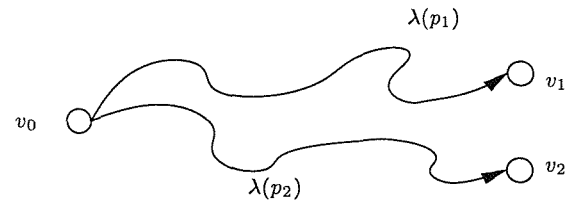


Figure 3.6: Summary labels for diverging paths

the extension of the summary is the union of the extensions of those expressions. For example, the expressions `Book` $\sqcup \exists$`AUTHOR.Jane-Austen` (books written by Jane Austen) and `Book` $\sqcup \exists$`AUTHOR.Charles-Dickens` (books written by Charles Dickens) have disjoint extensions, but their summary using $\sqcup$ is not a contradiction, having an extension which is the union of the two component extensions.

The summary operator may also be used to combine diverging paths, so as to represent all of the routes which may be taken from the initial node. For example, in Figure 3.6, the summary of the paths is $\lambda(p_1) \sqcup \lambda(p_2)$.

3.6 Summary

In this chapter we have presented a model of a query routing distributed search system which can be used to study the effects of different distribution topologies on system performance. Although there are a large number of deployed query routing systems, there have been only limited analytical studies of their behaviour to date. Our model draws on previously published results from the information retrieval, artificial intelligence and graph theory communities (in the form of the set-theoretic modelling of the information retrieval process, description and fuzzy logics and the path calculus of Cormen et al. (1990)), but the combination and application of these results to be a novel one.

In the next chapter, we will use this model to study the effects of different forward knowledge network topologies on the retrieval efficiency and scalability of a system (as given by the complexity measures in Section 3.3.5).

Chapter 4

Query Routing and Network Topology

4.1 Introduction

Using the model defined in the previous chapter, we can now investigate the effects of the distribution of forward knowledge (the topology of the forward knowledge network) on the efficiency of processing queries. An earlier version of the material appearing in this chapter was published in (Gibbins and Hall, 2001), and a summary of the complexities of the topologies discussed in this chapter is given in Table 4.1.

4.2 Forward Knowledge Distribution

The organisation of forward knowledge within a query routing search system plays a great part in the effectiveness, efficiency and scalability of the system. Forward knowledge should direct a query to the relevant servers with less effort than if the query were sent to all servers (an exhaustive search), and ideally the effort expended should grow more slowly than the number of servers as the system is expanded.

The order implicit in this network of forward knowledge is also important; the majority of existing query routing search systems presuppose the servers to have been arranged in a hierarchical manner with an omniscient root server (or group of servers) indirectly aware of the contents of every other server. This has been demonstrated to scale well in systems such as the Domain Name System (Mockapetris, 1987a), but it requires that a degree of control be exerted over the servers to force them into a hierarchy. Thus, ordered systems have the drawback that it may be costly or politically inexpedient to organise such a system; scalable ordered

systems rely on some centralised components for their operation, and the responsibility for running these may be too large to entrust them to an 'ordinary user' (the administrator of a simple leaf server, for example). These administrative, political or social costs are difficult to quantify (if not impossible), and are not reflected in the complexities of the given topologies.

A different approach treats all servers as peers and models the interactions between them as if they were social acquaintances; the majority of servers know only about their close neighbours, but a few servers have knowledge of more distant servers. This type of graph, known as a *small world network*, has attracted much attention of late in domains as diverse as paper citations, hypertextual linking on the World Wide Web (Albert et al., 1999), disease epidemiology and neural networks.

In both types of system, the key aim is that a properly labelled path exists from the server by which a query enters the graph, to the server or servers which can satisfy that query. With ordered network topologies it is possible to provide simple rules by which forward knowledge is passed, whereas disordered networks benefit from flooding techniques akin to those used by conventional routing algorithms.

In the example topologies given in the following two sections, the assumption has been made that all forward knowledge is faithful and complete, and this has affected the analyses of system performance accordingly, most of all those for query message traffic. While it is not possible to give specifics of performance for all systems in which forward knowledge is incomplete or unfaithful, a rule of thumb is that incomplete knowledge will reduce the query message traffic (as fewer referrals are generated) while unfaithful knowledge will increase the traffic. Similarly, the topologies studied have also been chosen such that a correctly labelled path will always exist between the entry points to the network and the other nodes, which ensures that all searches will have a recall of one (a relevant server can always be found by following the forward knowledge).

4.3 Ordered Networks

The majority of query routing search systems arrange their servers in a hierarchical fashion, with the data servers which hold the objects of the search at the leaves of the tree, and progressive layers of index servers further up the tree.

4.3.1 Single index server

This simplest hierarchical network for query routing consists of a set of data servers which hold the records in the system and a single index server which contains a complete description of the contents of each data server (as shown in Figure 4.1). Queries are submitted to the index server, which uses its forward knowledge to propagate the query to the relevant data servers. This arrangement is used by Napster (Section 2.2.13) and most Internet meta-search engines.

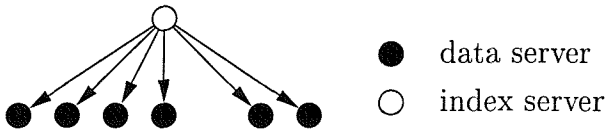


Figure 4.1: Single index server

We model this system as a directed graph $\langle V, E \rangle$ in which there is a distinguished $v_0 \in V$ which is the single index server and all edges in E are of the form $\langle v_0, v \rangle$. The contents of the data servers (the elements of the set $V \setminus \{v_0\}$) are denoted by the vertex labelling $\nu : V \to C$. Each data server passes a summary of its contents to the index server, so $\forall v \in V, v \neq v_0, \lambda(\langle v_0, v \rangle) = \nu(v)$. Therefore, given a query $q \in C$ which can be satisfied by the contents of a data server, there exists a path (edge) from the index server to that data server whose label also satisfies q.

Our estimates of the efficiency of the system rely on faithful and complete forward knowledge, and can therefore be regarded as best cases. The worst cases, those where all forward knowledge is entirely incomplete or unfaithful, are degenerate cases where the system reduces to either exhaustive search (unfaithful knowledge) or a null search in which no results are returned (incomplete knowledge).

When a query which may be satisfied by a single data server is submitted to the system, the generated query message traffic is constant with increasing system size (one request and response with the index server and the chosen server), as is the running time. Although query complexity is low, the index server is a bottleneck, since all queries presented to the system must be processed by it. The number of control messages sent initially scales as $O(|V|)$, the number of servers in the system. The number of entries in the routing table held by the index server also scales as $O(|V|)$. The update complexity for a single data server is constant for both time and communication (a single message sent to the index server).

4.3.2 Distinguished index servers

Although the system above is capable of routing queries to relevant data servers, it is unlikely to scale well as the number of data servers increases. The limiting factors are most likely to be the number of forward knowledge expressions that the index server must hold and the uneven load distribution that it is under.

It is natural to expand the system to a multi-layer hierarchy in order to address the first of these concerns. The lowest layer of the system consists purely of data servers which pass forward knowledge up the tree to index servers that summarise it and pass it in turn to their parent. Clients present their queries to the system as a whole by sending them to the root index server, which forwards the query to those second-level servers that might be able to pass it on to appropriate data servers (and so on, until the query reaches the data servers). To begin, consider the system shown in Figure 4.2, in which the leaves of the tree alone hold data and the servers internal to the tree summarise this data.

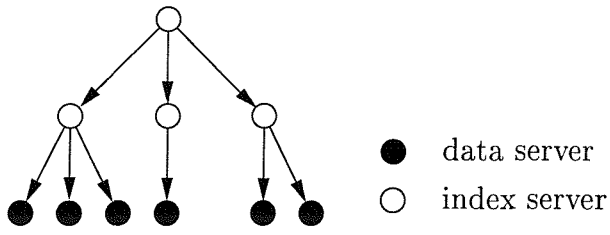


Figure 4.2: Hierarchy of distinguished index servers

Let $G = \langle V, E \rangle$ be a directed layered graph such that the set V of vertices is partitioned into the sets $L_0, L_1, \ldots, L_n$ and all edges $e \in E$ are of the form $\langle v, w \rangle, v \in L_i, w \in L_{i+1}$. The set L_0 contains the root elements of the graph; for this system we take L_0 to be singleton, so the graph is single-rooted. We denote the root vertex by v_0. For convenience we define the function $ch : V \to 2^V$ which maps a vertex onto its children:

$$ch(v) = \{x \in V : \langle v, x \rangle \in E\}$$

The vertices in V are labelled with the function $\nu : V \to C$. This describes the contents of the data servers in L_n with the most specific generalisation of those contents and describes all index servers as having no content.

$$\nu(v) = \begin{cases} \text{summary of contents of } v & \text{if } v \in L_n \\ \bot & \text{otherwise} \end{cases}$$

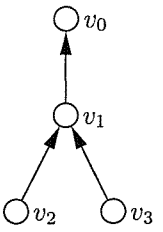


Figure 4.3: Constructing path labels

As in the previous case, the edges are labelled with the function $\lambda : E \to C$. The function differs from its previous definition by adding a summary of the edge labels from the layer below.

$$\lambda(u, v) = \nu(v) \sqcup \bigsqcup_{x \in ch(v)} \lambda(v, x) \tag{4.1}$$

If we take a query $q \in C$ for which some data server $v \in L_n$ contains a relevant document (which we can tell if $q \sqsubseteq \nu(v)$), there will be a chain of forward knowledge from the root server to this data server which will satisfy q if $q \sqsubseteq \lambda(v_0 \rightsquigarrow v)$. Consider the example shown in Figure 4.3. If v_2 contains some document relevant to q, that is if $q \sqsubseteq \nu(v_2)$, we can show that the label on the path $v_0 \rightsquigarrow v_2$ satisfies q (i.e. v_2 is reachable from v_0 given q, or $q \sqsubseteq \lambda(v_0 \rightsquigarrow v_2)$).

The path label is the most general label that will satisfy all of the edge labels along the path; if q is satisfied by this it will by definition be satisfied by the edge labels, guaranteeing the reachability of v_2.

When a query which may be satisfied by a single data server is submitted to the system, the generated query message traffic and query running time scale as $O(\log |V|)$, the depth of the hierarchy. The number of control messages sent initially scales as $O(|V|)$, as each data and index server passes forward knowledge towards the root while the control running time scales as $O(\log |V|)$. The mean number of entries in the routing table held by each index server, including the root, is constant with increasing system size (assuming an even distribution of servers beneath each index server), being equal to the breadth b of the hierarchy (the number of direct children of a node). If a single data server wishes to send an update, the complexity of this update is $O(\log |V|)$ messages over $O(\log |V|)$ rounds (considering a system which operates in synchronous parallel rounds, with one round being the time for each entity in the network to communicate with one or more of its neighbours, as discussed in Section 3.3.5).

4.3.3 Non-distinguished index servers

The model in the previous section may be expanded by weakening the distinction made between index servers and data servers. In the example system shown in Figure 4.4, all of the servers contain some data of their own, even those which contain forward knowledge about others.

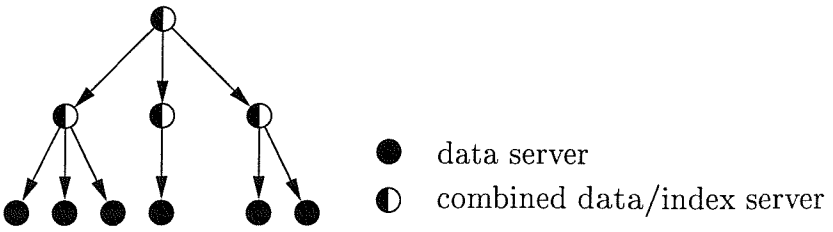


Figure 4.4: Hierarchy of index servers

The vertex-labelling function ν will now potentially return a non-bottom value for all $v \in V$, but this will not alter the definition of λ given in (4.1).

The efficiency estimate for this class of system is based on the worst case where the data server which contains relevant records is in L_n, so is equivalent to that for the system with distinguished index servers.

4.3.4 Multiple hierarchies

A further relaxation of this model may be made by allowing the hierarchy to have more than one root, as in Figure 4.5. We can model this by allowing the set L_0 to contain more than one vertex, but we must add a further restriction to the graph to ensure that all roots are functionally equivalent in terms of the queries which they can answer, or the data servers which can be reached from them. We define the *path projection* of a vertex v_0 as a subset of L_n, $\{v \in L_n : v_0 \rightsquigarrow v\}$, which contains those vertices which are connected to v_0. If data and index servers are distinct, two root servers are equivalent if their path projections are the same.

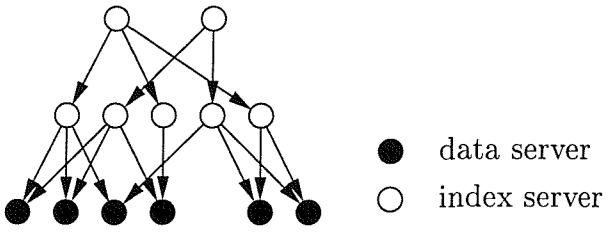


Figure 4.5: Multiple hierarchies

A graph with multiple hierarchies in which the roots are equivalent may therefore be considered as the conflation of two or more different hierarchies constructed atop the data servers in L_n. The data servers are reachable from any root, as for a single hierarchy with distinguished index servers.

This is no longer the case if the data servers and index servers are not distinct. Each hybrid data/index server contains data of its own, and so must be reachable from each root if queries are to have the same effectiveness regardless of where they are started, but may appear in different positions within the hierarchies beneath each root. However, if a given index server appears in different positions in several hierarchies, it must have a different set of forward knowledge for each hierarchy (the forward knowledge being for those servers which are beneath it in each hierarchy), and more importantly, it must have some way of knowing which set of forward knowledge is the correct set to use for a given query (i.e. it must have some idea of the hierarchical context in which it was asked the query). If this is not so, all of the hierarchies must be identical.

Ignoring this degenerate case, the efficiency of this class of system will be as that for single hierarchies with distinguished index servers.

4.3.5 Search expansion

So far, the hierarchical topologies which have been described are intended to be searched from the root to the leaves, but this often involves the expenditure of more effort than is necessary if all of the relevant servers lie within a particular sub-tree of the graph. In this case, searching from the root of this sub-tree is sufficient to retrieve all relevant objects.

Also, a single root represents a considerable bottleneck if it must process every query in the system. In order to spread the load more evenly across the servers in the system, queries are initially directed at a server chosen at random. If the query does not lie within the area of expertise of this server, it may choose to issue a referral to the server to which it customarily passes its forward knowledge (in RWHOIS terms, a *punt referral*).

The justification for such search expanding referrals differs from that of conventional referrals in that a server placed higher up in the hierarchy will be able to provide referrals that are relevant to a wider variety of queries. In the Domain Name System, a query (name lookup) that is addressed to a server (nameserver) which cannot provide a referral that narrows the search space (i.e. a referral to a

server which is better placed to answer the question) is passed up the tree until it reaches a server which can narrow the search space.

Search expanding referrals may also sometimes be generated by servers which are themselves able to generate referrals down the tree. These referrals allow the client to ask the query of a broader range of servers in order to achieve a better recall. The assumption made here is that a group of servers who all pass their forward knowledge to the same server are in some sense clustered, so that they have similar competences and areas of knowledge. A referral which directs the client to look further up the hierarchy is informing it that the parent server "*is likely to know about other relevant servers, because it knows about me*".

The knowledge which drives these referrals can be represented by another edge labelling, related to, but separate from the λ-labelling already in use. More specifically, this labelling (which we will denote by κ) is a labelling of the transpose of the original graph, G^T, in which all edges have a sense which is the reverse of that in G. The values taken by κ indicate the manner in which the search space is expanded (that is, the manner in which referrals which point up the hierarchy are generated). This may have a substantial effect on the performance of the system because it leads to the generation of extra messages, which affects the query complexity.

The systems listed in Section 2.2 which allow search expansion show some variety in the labellings they give to the transpose graph. WHOIS++ labels all transpose edges with the vertex labels at their source (equation (4.2)), in effect giving each data server v the reasoning that, if the index server u knows about me, it must also know about other things like me.

$$\kappa(v,u) = \nu(v) \sqcup \bigsqcup_{x \in ch(u)} \lambda(v,x) \tag{4.2}$$

DNS and X.500 take a different approach, in that they have a rigid naming hierarchy in place which is used to direct search. In this case the reasoning is that the parent of a server knows about a set of expressions that includes, but is not limited to, the knowledge of that server (equation (4.3)).

$$\kappa(v,u) \sqsupseteq \nu(v) \sqcup \bigsqcup_{x \in ch(u)} \lambda(v,x) \tag{4.3}$$

Both of these approaches use only a crude approximation to the total knowledge of parent servers because they have minimised the number of control messages which were sent around the system when the forward knowledge mesh was being built up.

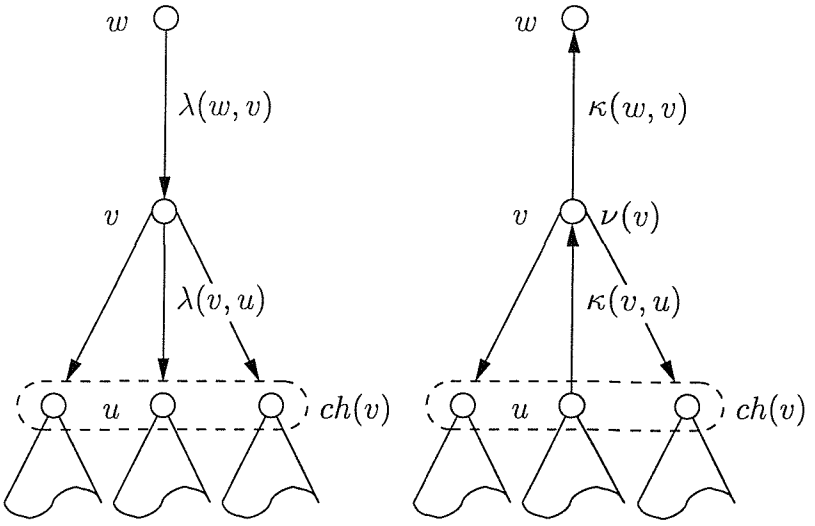


Figure 4.6: Expansion labels (κ)

If we allow knowledge about the contents and capabilities of servers to flow in reverse (from the root of the graph towards the leaves) as well in the conventional way, we can obtain a more accurate estimate for the total knowledge of a parent server (equation (4.4), see also figure 4.6).

$$\kappa(v,u) = \kappa(w,v) \sqcup \nu(v) \sqcup \bigsqcup_{x \in ch(v) \setminus \{u\}} \lambda(v,x) \tag{4.4}$$

Estimating the efficiency of this class of systems is more complicated than in previous cases due to the uncertainty of the initial server queried. In the worst case, the initial server is in L_n and has $v_0 \in L_0$ as its only ancestor in common with the target data server, which would give query message traffic of $O(\log |V|)$ messages because the query is propagated all the way to the root before it begins to be constrained. In the cases described by (4.2), (4.3) and (4.4), the routing table is of constant size, but the control message traffic in (4.4) scales as $O(|V|)$ because forward knowledge is propagated down the hierarchy as well as up. However, because routing knowledge is passed down the hierarchy as well as up, the complexity of propagating an update from a single data server is now $O(|V|)$ messages over $O(\log |V|)$ rounds (as described in Section 3.3.5) because the change in forward knowledge is propagated to all servers.

4.3.6 Complete Graph

This ordered graph is not a hierarchy, but describes a system with mutual forward knowledge, where all the servers know about the contents of all the other servers (see Figure 4.7). The message traffic generated by queries in this system is constant (4 messages) and takes constant time to process. Each server requires a routing table of size $O(|V|)$ and the control message traffic scales as $O(|V|^2)$ because the graph contains $|V|(|V|-1)$ edges. The running time for control messages is constant (one round). The complexity of propagating an update from a single server is $O(|V|)$ messages and takes constant time.

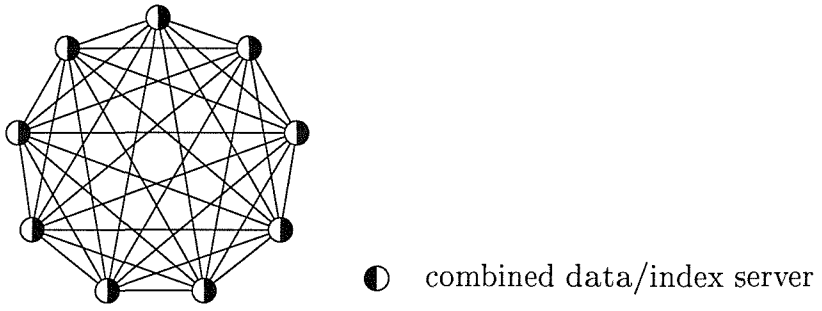
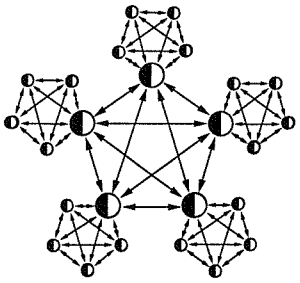


Figure 4.7: Complete graph

4.3.7 Councils

Councils are a cross between complete networks and hierarchies that were introduced in Lejter and Dean (1996) (Figure 4.8). The servers are divided into $|V|/b$ groups of b members each. Within a group, there is complete forward knowledge; every server knows about the contents of every other server. Each group is represented in a council by a chosen member from that group. This pattern is repeated in the councils; a council is also fully connected and makes representation to a higher level council, and so on.

Provided that $b \ll |V|$, a council has the same control complexity for queries, control messages and updates as a hierarchy. As b grows, the effort required to exchange control messages with group peers grows and the control message complexity approaches $O(|V|^2)$, that of a complete network (the running time is $O(\log|V|)$ regardless of b). The query message complexity and running time for a council are $O(\log|V|)$). For each group in which a server participates, it will have a constant b entries in its routing table. In the worst case a server will be a representative in groups at all levels; here the server's routing table size will scale as $O(\log|V|)$.



◐ combined data/index server

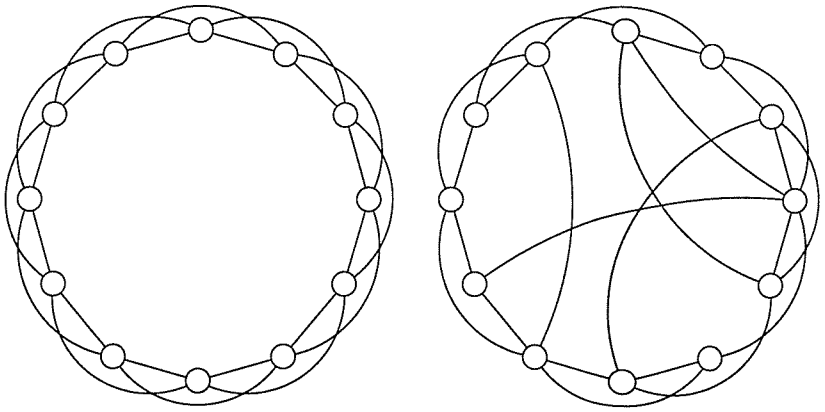
Figure 4.8: Councils

Also, because such a server handles a disproportionately large number of messages (being responsible for gatewaying messages between subtrees), it is a bottleneck, though not to as great an extent as the root server in a hierarchical network (here, the load is shared with its peer servers).

4.4 Disordered Networks

While ordered, hierarchical networks are a simple abstraction for studying the behaviour of query routing systems, it should be noted that similar real world systems are, for the most part, not ordered. In order to study networks with little or no implicit order, we need a family of graphs which are structurally similar to the types of disordered network that are likely to be encountered in real world systems, and for which it is known how the key properties which affect complexity and scalability change with graph size (such key properties being the diameter of a graph and the maximum vertex degree).

Small world networks are a class of random graphs which show a large degree of local order while retaining many of the characteristics of Erdös-Rényi random graphs (see Bollobás (1985) for a review of this work), such as low graph diameter (scales as $\log |V|$). In the model proposed by Watts and Strogatz (1998), the graph is based on the ring lattice $L_{n,k}$ (also known as a circulant graph or a 1-lattice), in which each of the n nodes in V are adjacent to their k nearest neighbours (for a total of $\frac{nk}{2}$ edges). This ordered graph used as a basis for the small world model could also be a rectangular lattice as used in Shehory (1999) (also known as 2-lattices with $k = 4$ in Watts (1999, p35)). This graph is then modified, with each edge being rewired at random with probability p. The resulting graph is greatly ordered locally, in that many neighbourhoods persist after rewiring, but appear more disordered



(a) Ring lattice $L_{12,4}$ **(b)** Modified ring lattice

Figure 4.9: Constructing small world networks

overall due to the existence of 'long distance' edges between otherwise distant nodes (see Figure 4.9 for an example).

These long distance edges have the effect of reducing the diameter of the graph. When their number is above some critical value, the diameter of the graph begins to scale as $\frac{\ln(n)}{\ln(k)}$, rather than as $\frac{n}{2k}$ in the unrewired ring lattice. This change in the nature of the graph to a small world network is characterised variously as a phase transition on the rewiring probability (Watts and Strogatz, 1998) or as a crossover phenomenon involving both the size of the network and the rewiring probability (Barthélémy and Amaral, 1999).

A number of real world systems have been shown to be small worlds (including the Gnutella peer-to-peer search system (Jovanovic et al., 2001)). However, the small world model has its flaws. Chief among these is its failure to model the probability distribution of vertex degree. A number of real world examples, including actor collaborations, webpage linking, electrical power grids and journal citation, follow a power-law distribution, where the probability of a vertex having a degree k is $p(k) \sim k^{-\gamma}$. In contrast, the small world model gives vertex degree a Poisson distribution. The practical upshot of this is that the real world networks contain a few exceptionally well connected vertices, while the vertices in small world networks are of more even degree.

Barabási and Albert (1999) call the class of random networks with power-law degree distribution *scale-free networks*, which differ from Erdös-Rényi random graphs and Watts-Strogatz small world networks in the method of their construction. Random graphs have a fixed number of vertices which stays constant throughout the

connecting process, while small world networks have not only a fixed number of vertices, but also a fixed number of edges by virtue of the rewiring process. Scale free networks are open, and grow by the addition of new vertices. These new vertices are connected to existing vertices, with a preference placed for links to vertices which are already well-connected. This rich-get-richer linking echoes the observed behaviour in real world networks (commonly cited papers attract more citations, for example) and results in a graph with a very few well-connected vertices, a slightly larger number of less well-connected vertices, and so on.

Scale-free networks still commonly have the small world property of a diameter which scales logarithmically as the number of vertices increases, as noted by (Albert et al., 1999). Barabási et al. (2000) give a model of scale-free networks which creates graphs such that $|V| = t+m_0$ and $|E| = tm$, where m_0 is the size of the initial 'seed' network, t the number of vertices subsequently added and m the number of edges which connect each new vertex to the graph. In Albert et al. (2000), the authors also note that scale free networks are also more robust than small world networks in the event of failure, if not in the event of malicious attack; the high degree vertices are crucial to the connectivity of the graph, but are comparatively rare making them less likely to fail if all vertices have an equal probability of failure, but allowing a malicious attack that specifically targets those vertices to have a greater effect.

Small world and scale free networks therefore offer an attractive abstraction of networks with partial disorder such as the World Wide Web (although see Broder et al. (2000) for a different characterisation of Web topology in which the Web as a whole does not possess small world connectivity). In particular, these graphs seem to be ideal for studying systems in which component entities are clustered by ability, interest or location (the limited communications afforded the cells in an amorphous computing system by Abelson et al. (1999) are a good example of the latter).

To begin with, we will assume that the servers in our search systems are not clustered by interest or ability, but form neighbourhoods based on some real world or network distance criteria. The edges in the graph show basic mutual awareness between servers. These edges may be concatenated using traditional network routing techniques to produce shortest paths to the other servers.

There is a fundamental difference between network routing and query routing, namely the treatment of addresses. In network routing, an address is a unique object used to identify a server. The routing process takes an address and constructs a path

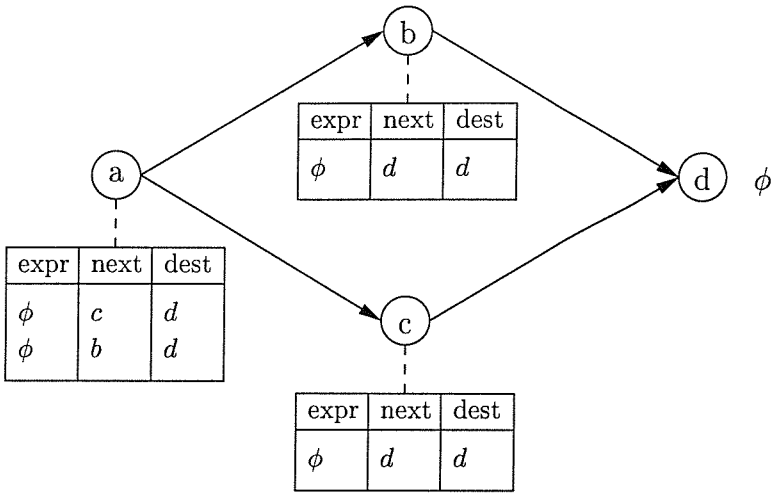


Figure 4.10: Augmented Routing Tables

from the source to the destination server which bears that address; each server's routing tables are indexed by these destination addresses.

In contrast, query routing takes a query expression, which does not uniquely identify a server, and attempts to construct paths from the source to each destination server which can satisfy the query expression. The routing tables may therefore contain more than one entry for a given expression, but this presents a problem while the routing tables are being constructed; do two entries with the same expression refer to the contents of different servers (and so should be retained as separate entries), or are they both references to the same server (and should be contracted into a single entry)?

This ambiguity may be resolved by including both the server's content summary and its name in the routing table, so that each routing table entry contains the expression which is to be compared with queries, the name of the server to which a referral should be generated and the name of the server which will eventually be reached if the referral is followed. An example of this is given in Figure 4.10, which shows a simple network of four servers. Server d contains some records whose summary we denote with ϕ. Servers b and c have routing table entries which indicate that received queries which are subsumed by ϕ should be routed to d. Server a has a routing table containing two entries, both for paths labelled with ϕ. These two paths ($a \rightarrow b \rightarrow d$ and $a \rightarrow c \rightarrow d$) both lead to d, but by different intermediaries. If a's routing table did not contain this information, a would be unable to distinguish between the two entries and would generate referrals to both b and c for queries matching ϕ, when only referrals to one or the other were required.

Making this change to the structure of routing tables brings to light a different question. If a server's routing table contains the name of a data server, why can't the query take a shortcut and be sent to the destination directly without having to traverse the path generated by the routing process? If the query is sent directly to the destination, none of the non-routing knowledge accumulated by the intermediate servers (namely, cached answers) may be brought to bear on the query.

4.4.1 Flooding

The first technique we present for searching in a disordered distributed system does not use query routing, and is including for comparison purposes. Gnutella (see Section 2.2.14) and FreeNet (see Section 2.2.15) are both peer-to-peer systems which do not use a centralised server to hold routing knowledge (and so could be considered to be 'true' distributed search systems), and both propagate queries using flooding or similar uninformed exhaustive techniques.

Although Gnutella clients flood a discovery message on joining the network, this is not used to gather or to distribute any routing knowledge, so the control message complexity and routing table size efficiency measures are not relevant. The complexity of querying a Gnutella system, however, is that of flooding the query message from the originating server to the rest of the system, which is $O(|E|)$ messages over a period of $O(\log |V|)$ ticks (the query is sent at most once over each edge in the network and reaches all vertices within a number of rounds equal to the diameter of the network).

FreeNet does not flood its queries as does Gnutella (effectively a breadth-first traversal of the system which uses delegation rather than referral), but performs a depth-first traversal of the system which reduces the peak loading caused by broadcasting a message to several destinations at the same time. This traversal is still uninformed, and there is no coordination between servers, so the query complexity is still $O(|E|)$, but spread over $O(\frac{|E|}{\log |V|})$ rounds. In practice, the control message traffic is less because FreeNet is a name resolution system in which any two entities that satisfy a query are considered equivalent; the search need not be exhaustive.

4.4.2 Distance Vector Routing

The Distance Vector (also known as Ford-Fulkerson or Bellman-Ford) algorithm is a dynamic, distributed routing algorithm widely used in the Internet, where it goes

by the name of the Routing Information Protocol (Hedrick, 1988). Each server maintains a routing table which contains, for each destination, the lengths of the shortest paths to that destination and the server to which data bound for that destination should be passed (see Figure 4.11).

Each server sends updates to its neighbours which consists of that server's current estimates of path lengths. Each server uses the received update to recalculate its routing table, substituting newly-found shorter paths for its current ones, if any exist. Over time, the routing tables converge to a stable state in which they contain optimal lengths. In this way the topological knowledge about the network is distributed amongst the servers in the network. Updates may be sent asynchronously, and as work by Bertsekas and Gallaher (1987) shows, the system will converge in finite time. Lynch (1996) notes that in a synchronous network, DV converges in $O(|V|)$ rounds with a communication complexity of $O(|V||E|)$, making this an expensive algorithm which scales poorly.

Each server's routing table requires storage of $O(|V|)$ to hold information about all other servers. When a query is submitted to the system, the query message traffic and running time scale as $O(\log |V|)$, the diameter of the graph. An update for a single server must be propagated using the standard DV algorithm, so its complexity is as that for the control traffic.

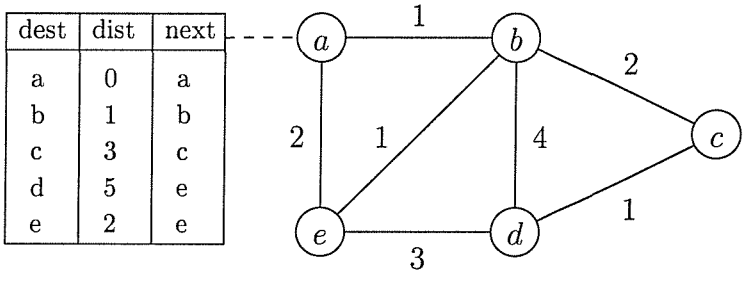


Figure 4.11: The routing table shown is for node a.

Distance Vector is considered to have some drawbacks, most notably the *count-to-infinity* problem where path length estimates rise indefinitely due to an inability to detect routing loops when link statuses fluctuate, but these are of less importance to us because we have restricted ourselves to fixed topology networks. One approach to the count-to-infinity problem is that used in the RIP implementation of DV, where path length estimates are given an upper limit of 16 hops. This avoids the count-to-infinity problem, but limits the use of the algorithm to networks with a graph diameter of 16 or less.

4.4.3 Path Vector Routing

Path Vector routing is derived from Distance Vector, and is designed to circumvent the count-to-infinity and routing loop problems. Each server's routing table contains a complete path to each of the listed destinations. This allows the servers to disregard routes which visit any server more than once, so removing routing loops. As a result, PV is not subject to the graph diameter limitation imposed on DV in order to avoid counts to infinity.

The main disadvantage of path vector routing lies in the size of the routing tables; while the tables in PV contain the same number of entries as those in DV, each entry contains a complete path of average length $\log |V|$ (the diameter of the graph). The communication complexity of PV is as that of DV.

4.4.4 Link State Routing

Open Shortest Path First (Moy, 1991) is a *Link State* algorithm, a dynamic routing algorithm based on Dijkstra's algorithm (summarised by Cormen et al. (1990, p.527)). This given in a form modified for use in a query routing system in Algorithm 4.1, following the notes on routing tables in Section 4.4. This differs from the cited version in the types of the arrays used to store the routing table (d and π in lines 2 and 3 for the distance and next columns) and the priority queue of vertices (Q, line 6), all of which use tuples containing a vertex and the content label for that vertex (given by the vertex labelling $\nu : V \to C$ - see Section 3.5.1) instead of just a vertex.

OSPF differs from DV in that each server has a complete copy of the topological knowledge about the network and calculates a routing table containing shortest paths for itself from this. The servers in OSPF exchange topological knowledge by using controlled flooding to send *link state advertisements* which describe their local topology to all other servers. The communications complexity of this all-nodes flooding operation is given by Harchol-Balter et al. (1999) as $O(|E| \log |V|)$, with convergence in $O(\log |V|)$ rounds (assuming a graph diameter that scales as $\log |V|$), but it is possible to use other algorithms that accomplish the same task with more amenable complexity (such as the Name-Dropper algorithm introduced in that paper, which has communications complexity of $O(|V| \log^2 |V|)$ and converges in $O(\log^2 |V|)$ rounds).

Each server has a copy of the topology of the network, which requires storage of at most $O(|V| + |E|)$, assuming a vertex adjacency representation. When a

Algorithm 4.1: Find the shortest paths to all vertices in a forward knowledge network from a single source

ADAPTED-DIJKSTRA($\langle V, E\rangle, \nu, w, s$)

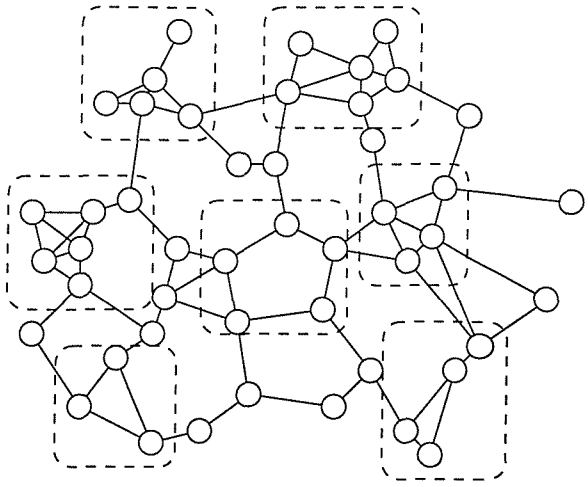
```
1  for all v ∈ V do
2     d[⟨v, ν(v)⟩] ← ∞
3     π[⟨v, ν(v)⟩] ← NIL
4  d[s] ← 0
5  S ← ∅
6  Q ← {⟨v, ν(v)⟩ : v ∈ V}
7  while Q ≠ ∅ do
8     pick u ∈ Q to minimise d[u]
9     S ← S ∪ {u}
10    for all v ∈ adj[u] do
11       if d[⟨v, ν(v)⟩] > d[⟨u, ν(u)⟩] + w(u, v) then
12          d[⟨v, ν(v)⟩] ← d[⟨u, ν(u)⟩] + w(u, v)
13          π[⟨v, ν(v)⟩] ← u
```

query is submitted to the system, the query complexity and running time scale as $O(\log |V|)$, the diameter of the graph. The flooding operation for a single server on update scales better than does DV, for a communication complexity of $O(|E|)$ in $O(\log |V|)$ rounds. If Name-Dropper is used, it should be noted that it was not designed for propagating messages from a single node, but rather for propagating messages from all nodes to all nodes; in this case, the flooding operation should be used to give a hybrid system in which global updates (when the system is started) use Name-Dropper, and individual updates use flooding.

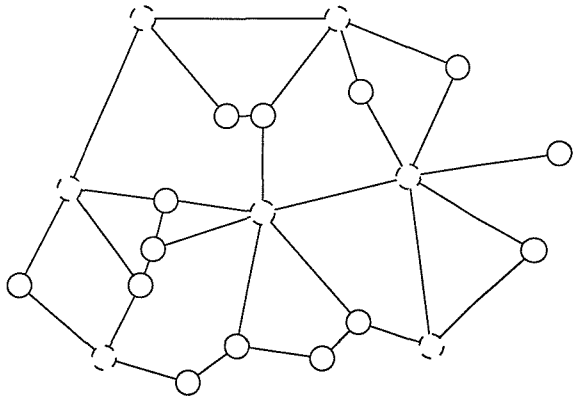
4.4.5 Hierarchical Routing

Link State and Distance Vector routing both have problems in very large networks. Distance Vector has a horizon to avoid counting to infinity, which places a limit on the size of network with which it may be used, and Link State requires that each server has the complete topology for the networks, which entails large routing tables and a potentially expensive flooding operation.

The routing table sizes can be reduced through the use of *hierarchical routing* (Lauder et al., 1991), which divides the network into a hierarchy of areas (sub-networks) such that areas at lower levels of the hierarchy are represented as single



(a) Before contraction



(b) After contraction

Figure 4.12: Hierarchical Routing

nodes at higher levels. Routing within an area is carried out as normal, but each area contains one server which acts as a gateway to the next higher layer in the network. In the layer above, routes are calculated to the gateway servers, but not to the servers beneath.

Hierarchical routing techniques are in use in the Internet; OSPF makes a distinction between internal routers which handle the traffic between networks in an area, and backbone routers (also known as gateways) which handle inter-area traffic.

We can model this by selecting connected components from the network and contracting them in place. Contracting such a component replaces it with a single vertex which is adjacent to all of the vertices to which the vertices in the component were originally adjacent, as in Figure 4.12. The topology of the resulting graph is that of the new higher layer of the network. On the Internet, the contracted

components are the networks which exist within organisations and companies, while the higher layer is the backbone network of connections which join the local networks together.

The vertices of the contracted component are represented within the higher layer by a member vertex (a gateway), chosen by some means (normally explicitly by the network designer, although this could perhaps be automated by using *leader election* algorithms, as discussed in Singh (1997) and Awerbuch (1987), which allow a group of communicating entities to select a single member from amongst themselves).

In the case of Watts-Strogatz networks, any neighbourhood in the unaltered ring lattice $L_{n,k}$ which contains at most $\frac{k}{2}+1$ vertices is a clique. Contracting the maximal such clique to a single vertex results in a new ring lattice, $L_{n-\frac{k}{2},k}$. The query message traffic remains as before, twice the diameter of the graph, but the routing table size may be dramatically reduced, depending on how deeply the hierarchy is constructed.

The control message traffic is also reduced, the exact amount depending on hierarchy depth again, but there is also a need for area leaders to advertise aggregate routes from the area to the layer above and vice versa, much as in Section 4.3.5.

4.5 Mutual State

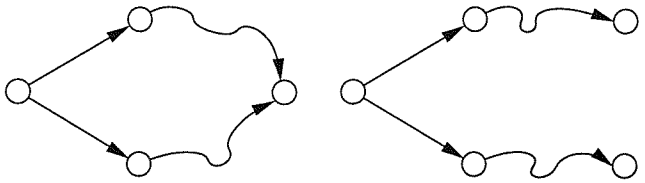
If the only forward knowledge in a system is direct (direct knowledge of the contents of another server, with no forward knowledge about the forward knowledge of other servers) almost all queries will terminate prematurely without finding any relevant servers, or *stall*, even though the system may contain relevant servers. This happens because there is no appropriately labelled path (eg. one which subsumes the query) which leads to the destination server, where results which will satisfy the query are to be found, and the necessary referrals have not been generated.

For such a path to exist with only direct knowledge, each server along the path would have to have summarised its contents in an unfaithful manner so that all of the summaries could be considered relevant to any query. If however it were the case that servers routinely expressed their contents using unfaithful summaries, there would be many more incorrect referrals where the client was directed to a server which did not hold relevant data, and the search would tend towards an exhaustive one.

In general, predicting the degree at which the phase transition between stalling and exhaustive search occurs is extremely difficult; the topology of the forward

Topology	Control traffic	Control time	Query traffic	Query time	Update traffic	Update time	Routing table
Ordered							
single	$O(\|V\|)$	constant	constant	constant	constant	constant	$O(\|V\|)$
hierarchy	$O(\|V\|)$	$O(\log \|V\|)$	$O(\log \|V\|)$	$O(\log \|V\|)$	$O(\log \|V\|)$	$O(\log \|V\|)$	constant
hierarchy w/ expansion	$O(\|V\|)$	$O(\log \|V\|)$	$O(\log \|V\|)$	$O(\log \|V\|)$	$O(\|V\|)$	$O(\log \|V\|)$	constant
complete	$O(\|V\|^2)$	constant	constant	constant	$O(\|V\|)$	constant	$O(\|V\|)$
council	$O(\|V\|)$	$O(\log \|V\|)$	$O(\log \|V\|)$	$O(\log \|V\|)$	$O(\log \|V\|)$	$O(\log \|V\|)$	constant
Disordered							
gnutella (flooding)	—	—	$O(\|E\|)$	$O(\log \|V\|)$	—	—	—
distance vector	$O(\|V\|\|E\|)$	$O(\|V\|)$	$O(\log \|V\|)$	$O(\log \|V\|)$	$O(\|V\|\|E\|)$	$O(\|V\|)$	$O(\|V\|)$
link state	$O(\|E\| \log \|V\|)$	$O(\log \|V\|)$	$O(\log \|V\|)$	$O(\log \|V\|)$	$O(\|E\|)$	$O(\log \|V\|)$	$O(\|V\|)$
link state w/ name-dropper	$O(\|V\| \log^2 \|V\|)$	$O(\log^2 \|V\|)$	$O(\log \|V\|)$	$O(\log \|V\|)$	$O(\|E\|)$	$O(\log \|V\|)$	$O(\|V\|)$

Table 4.1: V is the set of vertices in the underlying graph (servers in the system), E is the set of edges



(a) Redundant **(b)** Not redundant

Figure 4.13: A group of referrals are redundant if they are the initial edges of a group of labelled paths which terminate at the same vertex.

knowledge graph, the edge labels, the queries and the server to which the query was initially presented all play a part.

Forward knowledge in a query routing system directs and constrains the search, allowing clients to prune irrelevant vertices from the referral graph, but aggregated forward knowledge weakens these constraints by generating many referrals to servers which in turn will generate referrals to the required data.

In the worst case, a query is directed to one of a group of servers which form a clique in the forward knowledge graph. If one of these servers has relevant data, a referral will be generated to it, but a referral will also be generated to every other server in the clique, even though they only point at the (known) relevant server.

To prevent referrals to known servers, the client should present each server with which it communicates with the query and the list of servers it has already visited (its *out-list*) so that the server will only generate referrals to previously unseen nodes. This in itself is not sufficient to constrain the unwanted referrals in the worst case above, since none of the nodes in the clique have yet been visited when the referral to the clique is accepted by the client.

In response, the referrals generated by the servers should not only include the name of the server to which the client is being referred, but also the names of the relevant destinations which may be reached by following this referral and the distance that the destinations lie from the referred-to server. This fulfils the notion of mutual state introduced in Section 3.4; clients tell servers of the current state of the search so that the servers do not create unnecessary referrals, and the servers provide enough of their routing knowledge to the clients for the clients to be able to make decisions on which referrals to act on (ie. which to add to their *in-list*) and which to discard as redundant.

A server may choose not to generate referrals, but to delegate the search instead by acting as a client with an empty in-list and the out-list given it by the client who

passed it the query. This, however, brings to light coordination problems between servers. Although this new server-client will not visit any servers already visited by the original client, its out-list will quickly diverge from that of the original client, so there may be duplication of effort between them.

4.6 Summary

In this chapter we have examined the effects of forward knowledge distribution on the efficiency of query routing search by using the model described in Chapter 3 to characterise a number of network topologies and to determine the complexity of querying and of maintaining the forward knowledge network. In Chapter 5 we will describe the design and implementation of Phyle, an agent framework which we have used to build agent systems that use query routing.

Chapter 5

The Phyle Agent Framework

5.1 Introduction

In this Chapter, we describe the design and implementation of an agent framework, which was developed solely as part of this research in order to allow us to construct agent-based query routing systems. Our decision to develop our simple agent framework, Phyle, was motivated by the need to demonstrate the behaviour of an agent-based query routing system (which necessarily requires that such a system be built). The main objectives which drove the design of Phyle were that it impose few restrictions with respect to the choice of agent communication and knowledge representation language and allow new performatives or protocols if necessary, so that it could be used to investigate the issues surrounding query routing agent systems.

At the time this work was undertaken (early 1998), there were very few frameworks for building agent systems available, and still fewer which made source code available. This situation has since been remedied with the growth of open source agent platforms such as FIPA-OS (Poslad et al., 2000) and JADE (Bellifemine et al., 1999). Unfortunately these appeared too late to be of use to this work, which made the development of Phyle necessary for this work to take place. In this chapter we describe the Phyle agent framework, concentrating on those parts of Phyle which differ from other agent systems. In particular, we describe the subsumption-based message handling techniques used (Section 5.2.3) and the accompanying algorithms for lattice manipulation (Section 5.2.4).

Finally, we describe Paraphyle, a simplified simulation environment for query routing agent systems.

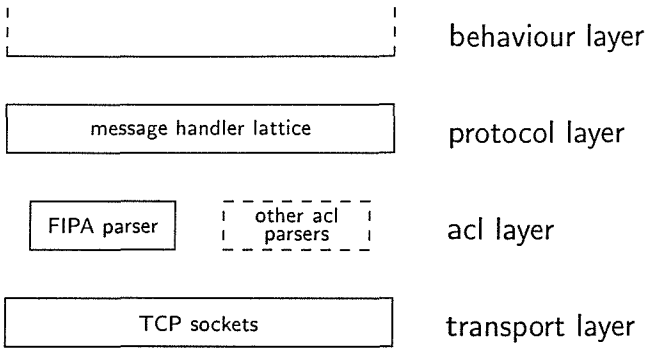


Figure 5.1: Conceptual layers in Phyle

5.2 The Phyle Agent Framework

Phyle was developed during 1998, and had the twin (and sometimes contradictory) design philosophies of simplicity and flexibility. As a framework for building agent-based systems, we felt that where possible, it should make as few assumptions as possible about the type of agent system which was to be built. The main assumption that was made concerned the likely complexity of the agents which would be built using the framework; for distributed information management tasks, we believe that:

- complex collective behaviour can arise from simple individual behaviours
- many agent tasks can be carried out without resort to sophisticated machine learning techniques

The design of Phyle owes much to Shoham's Agent-0 language (Shoham, 1993), adopting a weak approach to agent communications by default, in which incoming messages trigger the execution of one or more actions from a library of canned behaviours. The selection of appropriate actions depends in part on the message received. Each canned behaviour carries one or more exemplar patterns describing the messages which will trigger it; an incoming message is compared with these patterns in order to determine which actions should be performed.

The selection of appropriate actions from the behaviour library depends not only on the messages received, but also on the context in which they are received. A message received as part of a conversation (the confirmation of a purchase order, for example) has a different meaning to an identical message received out of context (which would simply make no sense if the recipient has not received the original purchase order).

Agents built using Phyle communicate at the transport level via BSD-style Internet sockets, for simplicity. Phyle has where possible reused existing standards for

agent communication; the agent communication language used by Phyle is the FIPA ACL (FIPA, 1997b), while the content language used is the FIPA knowledge representation language, SL0. Although Phyle deals natively with the FIPA languages, it does so in as neutral a fashion as possible. It has been possible to write naïve conversion routines which transform simple KQML, KIF and Prolog expressions into their rough FIPA equivalents, so allowing a limited degree of communication with agents that use these languages. Phyle has been designed in a layered manner, as shown in Figure 5.1, in order that new communication mechanisms (for example, a different transport layer or ACL) may be added without undue difficulty.

5.2.1 Agent Naming

An important component of an agent framework is the naming system by which agents identify themselves to each other, but the form which agent names take must necessarily depend on the means by which agents communicate. In systems such as CORBA (OMG, 1996) or FIPA (FIPA, 1997a), inter-agent messages are mediated by a common communications channel. The communications channel (called an *Object Request Broker* or *ORB* in CORBA and an *Agent Communications Channel* or *ACC* in FIPA) routes messages between agents, and each agent is situated on a channel (see Section 2.3.2).

In systems which contain only a single channel, messages sent between agents are routed locally (Figure 5.2(a)), so in this instance an agent's address need only be unique within the scope of the shared communications channel (even if this is not generally the case for multi-channel systems). In systems which contain more than one channel, messages sent between agents situated on different channels must be passed from one channel to the other (Figure 5.2(b)), which requires either that the channel on which an agent is situated is encoded in its address, or that each channel maintains an address book which lists agents by name and the channels which they each use. Both CORBA and FIPA adopt the former approach, with agent names containing the communications channel on which the agent is situated as well as a local name component which is unique within that channel.

Although we have chosen to use the FIPA Agent Communication Language, Phyle is not strictly FIPA-compliant because we have not implemented the CORBA-like ACC. We have chosen instead to make agents communicate directly with each other by sending FIPA ACL messages over BSD-style Internet sockets, rather than by talking to an ACC as is required by the FIPA specifications. We are not alone

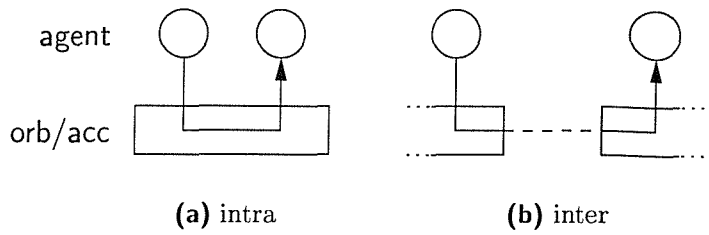


Figure 5.2: Bus Communications

in making this choice, as demonstrated by British Telecom's Zeus agent system (Nwana et al., 1999), which also chooses to transmit FIPA messages over Internet sockets.

Because Phyle agents communicate in a point-to-point fashion without the use of intermediaries, the naming scheme used must be such that an agent is able to resolve a name to an address which can be used to open a channel for communication. We have adopted a simple naming scheme which uses a transparent URL schema that identifies the machine on which an agent resides, the port on which it listens and a user-determined name (see Figure 5.3).

ans://*hostname*:*port*/*agent-name*

Figure 5.3: Agent Naming System URL Schema

Although it incurs some overhead in the opening and closing of TCP connections, we have implemented Phyle's point-to-point communications in such a way that a new connection is opened for each message which is sent. Either the sender or the receiver may close the connection when the communication is finished (although in practice, the sender usually closes it).

5.2.2 Agent Environment

Agent systems commonly use a facilitation service which enables an agent to locate other agents which are able to help it achieve its goals, normally by providing some service. The facilitation service is a specialised middle agent (Decker et al., 1996) with which agents may advertise their services or enquire about agents which provide some service. The facilitation service may be provided by a single agent, or by a federation of agents which share advertisements.

However, if the facilitator's ability to mediate service requests is treated as a service, we are rapidly left with a chicken and egg problem whereby an agent needs a facilitator to find a facilitator!

There are two main approaches taken to solve this bootstrap problem, both straightforward, but both with drawbacks. One approach stipulates that a facilitator is created with a known name which is provided to all other agents when they are initialised. This is a low effort way of ensuring that all agents have the necessary a priori knowledge required to locate the facilitator, and from there the other agents in the system. The FIPA agent standard takes this approach, and requires that each Agent Communications Channel has a *directory facilitator* (see Section 2.3.2). In this case, the facilitation is provided locally, so that an agent need look no further than its local environment, but this not need be so; we can envisage a system in which facilitators are shared between agent execution environments, albeit a brittle one (the failures of one agent execution environment would render any others which depended on its facilitator inoperable).

The second approach to the bootstrap problem uses broadcast messages; when an agent needs the services of a facilitator (and doesn't know where to find one), it transmits a broadcast request message. On receipt of this message, the facilitator informs the agents of its presence and proceeds to communicate directly with the agents. The service location request may be processed in one step (the broadcast message contains the query, which the facilitator replies to directly) or in two steps (the broadcast message is used only to locate the facilitator, which then waits to be contacted by the agent with the service request itself).

This approach is used by the lookup service in the Sun Jini system (Sun, 2000). In this system, agents repeatedly send multicast UDP packets containing their request, with increasing times-to-live. When the Jini facilitator (known as a *djinn*) receives such a packet, it opens a TCP channel to the agent and sends the response (the location of the service provider) to the agent.

In Phyle, we have chosen to use the former approach for simplicity's sake; the broadcast approach is best suited to systems with a common channel which mediates all communications (such and the CORBA ORB or the FIPA ACC), which we do not have. Therefore, all Phyle agents expect to be able to find a facilitator listening on port 4000 on the machine on which they are running.

As an aside, this problem of locating a agent which can perform some service is essentially the same problem that we're trying to solve using query routing. Given a starting point (an initial server to query), a query routing system locates an agent which provides a certain service (in our case, the ability to answer certain types of question) by using a series of mediators. When queried about service providers, a facilitator's response can be viewed like a query routing referral (and conversely,

locating a starting point for a query routing search is akin to locating a facilitator prior to sending a service location request).

5.2.3 Message Handling

As described in Section 5.2, a Phyle agent contains a library of predefined behaviours which are invoked in response to a received message or an environmental event. On receipt of a message, an agent must identify which particular canned behaviour (if any) should be initiated as a result of that message; each behaviour is contained in a message handler which has an attached pattern which will match those messages (using a structural matching technique such as unification) which can trigger the behaviour.

This much should be quite familiar to those familiar with the message passing style of object-oriented programming. However, we also remove the restriction that a message may trigger only one message handler (as would be the case in an OO system of the method invocation style); a message sent to a Phyle agent may trigger any number of message handlers. The process of determining which handlers should be triggered becomes more complicated because we also require some way of selecting those handlers which we deem to be most appropriate to the message.

For example, responding to certain specific queries may require more intensive methods than more general queries. We don't wish to waste effort by applying the intensive methods to queries which can be answered more simply, so a well-written agent should apply the intensive methods only when appropriate.

We take the most appropriate handlers to be those which most closely subsume a message; the message handlers may therefore be ordered by subsumption into a lattice to reduce the effort required to search the library of handlers (see Section 5.2.4 for more details of this). An agent's library of handlers is not a static structure. Handlers may be inserted and removed as the agent's needs change. In general, the lattice contains two types of entry:

- permanent entries, representing an agent's advertised capabilities (those messages which an agent has explicitly declared itself to be able to handle)
- temporary entries, representing an agent's expected communications from other agents (for example, if an agent is awaiting a response to a query it has sent, there will be an entry for the answer)

The use of temporary entries in the lattice allows us to model agent communication protocols like those used by FIPA (FIPA, 1997b); an agent sending a `query-ref`

message to another agent would create temporary handlers in anticipation of both a successful outcome (an `inform` message containing an answer) and the various failure modes (`failure`, `refuse` or `not-understood`). When a message matching any of these is received, the corresponding behaviour is invoked and then all the handlers in this group are removed from the lattice.

In the general case, each communication protocol is represented with a finite state machine, each of whose transitions is labelled with a message pattern, a set of handlers to add to the lattice and a set to be removed. An example FSM for the FIPA request protocol is shown in Figure 5.4. Although common sense suggests that the obsolete message handlers should be removed on leaving the old state and the new handlers added when entering the new state, this gives rise to a potential race condition where an agent may fail to recognise a legitimate message for the protocol because it has yet to create the handlers for that message. When a protocol sequence is initiated, a new FSM is created from an exemplar (the collection of protocol FSM exemplars held by an agent is the library of protocols in which it can participate), and when completed, it is destroyed.

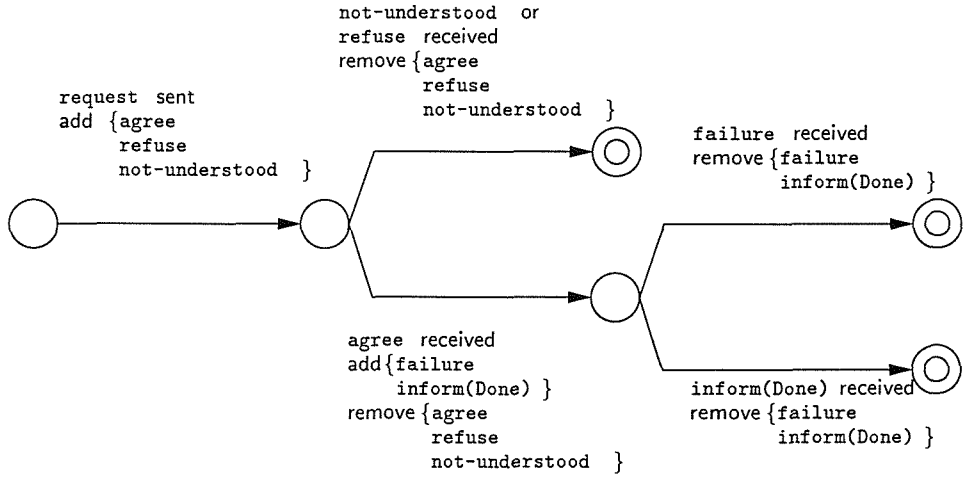


Figure 5.4: Finite state machine for FIPA Request protocol

The use of a single data structure to hold the transition labels for all of the current protocol FSMs in an agent makes this a different approach to that taken by other agent systems such as Zeus (Nwana et al., 1999), where there is no single structure containing all of an agent's communicative expectations.

5.2.4 Subsumption Lattices

In the previous section, we described the message handling component of Phyle, which uses a lattice to order and select the message handlers, but we refrained from describing the lattice itself. The intention behind our use of lattices is that we may be able to determine the set of message handlers which most closely relate to a received message without recourse to an exhaustive comparison with all the agent's message handlers. To this end, we construct the Hasse diagram for the lattice and use the partial ordering on the message handlers to constrain a search through the lattice.

Conventional lattice theory has concentrated on fixed orderings over fixed sets, but our design for a lattice-based message handling component requires that we be able to add and remove handlers from the lattice. *Formal concept analysis* (Ganter and Wille, 1998) is an offshoot of lattice theory which deals with the analysis of data, in particular with the abstraction of the data to form *concepts* which describe groups of related objects. This field has produced some results which relate to our problem, namely the incremental creation of lattices (Godin et al., 1995), but these are not directly applicable for two reasons.

Firstly, the lattices used in formal concept analysis are *Galois lattices*, in which each concept (element) in the lattice is a tuple composed of the extension of the concept (the set of individuals in the concept) and its intension (the set of common features possessed by members of the concept). In our system, we cannot realistically enumerate the extension of a given expression (the messages which it matches). Similarly, while many parts of a message envelope can be viewed as boolean features in the intension of the expression, the languages in which the message content is written are too expressive for content sentences to be reduced to such a feature set. Secondly, formal concept analysis concentrates on the construction of lattices, whereas we also need to be able to remove elements from our message handler lattices.

In the description of our algorithms, we take a subsumption lattice to consist of a partial ordering given by $\sqsupseteq$ over a set A. The functions $children : A \rightarrow 2^A$ and $parents : A \rightarrow 2^A$ (5.1, 5.2), which give the upper and lower covers for elements of A, define the edges in the corresponding Hasse diagram. If there is an edge from x to y and $x \sqsupseteq y$, then $x \in parents(y)$ and $y \in children(x)$.

$$x \in children(y) \iff y \sqsupseteq x \wedge \nexists z \in A.(z \neq x \wedge y \sqsupseteq z \sqsupseteq x) \quad (5.1)$$

$$x \in parents(y) \iff x \sqsupseteq y \wedge \nexists z \in A.(z \neq x \wedge x \sqsupseteq z \sqsupseteq y) \quad (5.2)$$

Algorithm 5.1 is used to locate the most appropriate message handlers in the lattice, that is the message handlers whose templates most closely subsume the received message. The algorithm starts with an initial set of candidate solutions, C, which contains the lattice supremum $\top$ only. The algorithm iterates with a while loop in line 3 which exits when all candidates have been examined. When a candidate is examined (lines 5-8), each of its children is checked to see if they subsume the message. If a child subsumes the message, it is added to the set of candidates and the parent (candidate) is removed. If all the children of a candidate do not subsume the message, the candidate is one of the most specific expressions to subsume the message and is added to the set of solutions, S (lines 9-11). Our description of this algorithm does not explicitly suppose it to be either a depth- or a breadth-first search; this is determined by the way in which candidate solutions are selected for testing in line 4.

Algorithm 5.1: Find the most specific subsuming expressions for x in the lattice

MOST-SPECIFIC-SUBSUMING(x)

```
1  S ← ∅
2  C ← {⊤}
3  while C ≠ ∅ do
4    pick y ∈ C
5    for all z ∈ children(y) do
6      if z ⊒ x then
7        C ← C \ {y}
8        C ← C ∪ {z}
9    if y ∈ C then
10     C ← C \ {y}
11     S ← S ∪ {y}
12 return S
```

If an agent's message handler groups are to be fluid, we must be able to insert and delete new message handlers in the lattice. Insertion of new expressions in a subsumption lattice is a two-step process, as given in Algorithms 5.2 and 5.3.

Algorithm 5.2 links the new expression to its parents, those expressions in the lattice which most directly subsume it. This proceeds as in Algorithm 5.1, with two differences. Firstly, the algorithm links the new expression as a side effect (lines 10-11), so there is no solution set to be accumulated and returned (cf. lines 1, 11 and 12 in Algorithm 5.1). Secondly, the condition in line 5 (cf. line 6 in Algorithm 5.1) uses strict subsumption $\sqsupset$ in order to preserve the ordering in the lattice.

Algorithm 5.2: Find parents of new expression x

LATTICE-FIND-PARENTS(x)

```
1  C ← {⊤}
2  while C ≠ ∅ do
3    pick y ∈ C
4    for all z ∈ children(y) do
5      if z ⊐ x then
6        C ← C \ {y}
7        C ← C ∪ {z}
8    if y ∈ C then
9      C ← C \ {y}
10     children(y) ← children(y) ∪ {x}
11     parents(x) ← parents(x) ∪ {y}
```

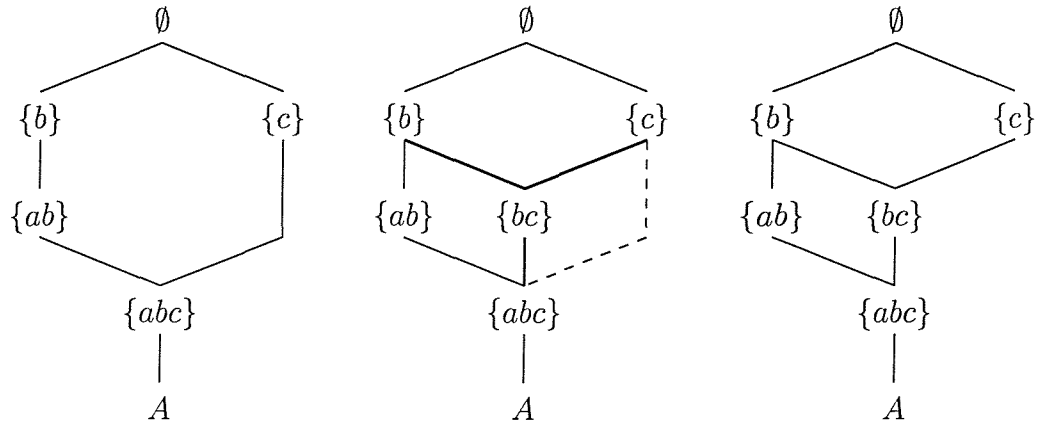
Algorithm 5.3 completes the addition of the new expression to the lattice by linking it to its children, and by unlinking those children from their old parents where necessary. The algorithm iterates over all of the siblings of the new expression (the children of the parents of the new expression, lines 1-2), and if any are strictly subsumed by the new expression (line 3), they are unlinked from those parents they have in common with the new expression (lines 4-5) and relinked beneath the new expression (lines 6-7).

The overall insertion operation is illustrated by an example in Figure 5.5. For simplicity, we represent the expressions in the lattice by sets (which could be construed to be their intensions), so subsumption is replaced by the subset relation and $\top$ and $\bot$ are replaced by $\emptyset$ and A, the universal set, respectively. The figure shows the Hasse diagrams before, during and after the insertion of the new expression $\{bc\}$ into the lattice. The expressions $\{b\}$ and $\{c\}$ are the parents of the new expression, while $\{abc\}$ is its child. The heavy links show new edges in the Hasse diagram, while the dashed lines are edges to be removed.

Algorithm 5.3: Find children of new expression x

LATTICE-FIND-CHILDREN(x)

```
1 for all y ∈ parents(x) do
2   for all z ∈ children(y) do
3     if x ⊐ z then
4       children(y) ← children(y) \ {z}
5       parents(z) ← parents(z) \ {y}
6       children(x) ← children(x) ∪ z
7       parents(z) ← parents(z) ∪ x
```



(a) Before **(b)** During **(c)** After

Figure 5.5: Lattice Insertion

Deleting an element from the lattice is the converse of the insertion operation. The links to the removed expression from its parents are removed and new links from the parents to the children of the expression are instated where necessary, then the links to the children of the expression are removed. An edge between a parent and a child is created only if there are no other intermediary nodes.

In Algorithm 5.4, lines 1-2 and 7-8 remove the links from the deleted node to its parents and children. Lines 3-6 iterate over the siblings of the deleted node (ie. its parent's children), creating links from those to the children of the deleted node where necessary.

Algorithm 5.4: Delete expression x from lattice

LATTICE-DELETE(x)

```
1 for all y ∈ parents(x) do
2   children(y) ← children(y) \ {x}
3   for all z ∈ children(y) do
4     for all w ∈ children(x) do
5       if w ⋢ z then
6         children(y) ← children(y) ∪ {w}
7 for all y ∈ children(x) do
8   parents(y) ← parents(y) \ {x}
```

5.3 The Paraphyle Simulator

In order to demonstrate the scalability of the different query routing topologies discussed in Chapter 3, we need to construct large systems containing hundreds of thousands of agents. Although the Phyle agent framework described above could be used to build such a system, each agent uses significant computing resources (a bare Phyle agent with no knowledge base and empty message lattice uses approximately 2Mb of RAM). With the limited computing facilities available, the most appropriate approach to take is one of simulation.

We have constructed a simulator for Phyle agents which we call Paraphyle. This abstracts and simplifies several aspects of query routing Phyle agents in order to make possible the simulation of large query routing systems. The aspects of Phyle which have been most radically altered are as follows:

Agent Communication Languages: The ACL component of Phyle, with its sophisticated protocols and flexible message handling, is unnecessary in a query routing simulation where the vast majority of messages will be of the query-referral/query-response variety.

Networking: A simulation of a distributed system need not be distributed itself. Indeed, for the purposes of data collection and monitoring, a non-distributed system presents none of the problems encountered in distributed systems (timestamp synchronisation, data collation, etc). Paraphyle therefore does not require networking support.

Knowledge representation: Phyle contains an implementation of the FIPA knowledge representation language SL which is used to express knowledge about information resources. In Paraphyle, information resources are represented

by identifiers only. Query and forward knowledge expressions are represented by their intensions, the set of characteristics possessed by the information resources which they describe. By doing this, the tests for subsumption and unification are reduced to tests for subset and set intersection respectively.

User interface: Paraphyle is designed for batch rather than interactive use, so the html-generating components of Phyle may be dispensed with.

5.4 Summary

In this chapter we have described the Phyle agent framework and the simplified Paraphyle simulator. In Chapter 6, we will describe the application of Phyle to the construction of a query routing agent system, while in Chapter 7 we use Paraphyle to simulate a much larger example of such a system.

Chapter 6

An Agent System for Query Routing Search

6.1 Introduction

In Chapter 5, we described Phyle, our FIPA-based agent framework. This description was deliberately application-neutral, because we wished to explain certain aspects of it without domain bias; we will now relate the contents of Chapter 5 to the work presented in Chapter 3, where we described the query routing problem. In this chapter, we describe the design of an agent-based query routing system which uses the Phyle agent framework, making note of the conversation protocols used by such a system, the manner in which forward knowledge is represented, the search algorithms used and ontologies for the application domains which are being searched.

6.2 Query Routing Protocols

The FIPA Agent Communication Language provides the means to describe stereotypical message exchanges, or protocols. The specification of the ACL (FIPA, 1997b) contains a formal model of the ACL which gives a clear semantics to the performatives used in terms of their necessary preconditions and their *perlocutionary effect* (the effect on the hearer, also referred to in FIPA terminology as the *rational effect*). Of particular interest is the basic directive, `request`, by which an agent indicates to another agent that it wishes an action be performed (the description of the action specifies who is to perform the action). The perlocutionary effect of this message is that the action is performed.

```
(query-ref :sender user
           :receiver lib-01
           :reply-with tn23
           :content (iota ?x (and (creator ?r "Austen, Jane")
                                  (identifier ?r (isbn ?x)))))

(inform :sender lib-01
        :receiver user
        :in-reply-to tn23
        :content (= (iota ?x (and (creator ?r "Austen, Jane")
                                  (identifier ?r (isbn ?x))))
                    ((0140430721
                      0140434259
                      0140430105
                      0140430059))))
```

Figure 6.1: FIPA `query-ref` and response

The FIPA `query-ref` message is a composite performative, a `request` that the receiver `inform` the sender of those expressions which satisfy a query expression. The perlocutionary effect of the `query-ref` message is that the receiver sends an `inform` message back to the sender. It is reasonable to suppose that the sending agent forms an expectation of this as a possible outcome, perhaps by adding an appropriate entry in its set of message handlers as in Section 5.2.3. The agent's expectation is not just for a performative, but also for a message envelope (it would expect to receive the answer from the agent it asked, for example) and for a message content of a particular type. In the example in Figure 6.1, the user agent sends a request to a library catalogue agent for the ISBNs of books written by Jane Austen. A literal interpretation of the message might be "tell me those identifiers which are ISBNs of resources whose creator is Jane Austen". Correspondingly, an interpretation of the answer might be "the following are identifiers which are ISBNs of resources whose creator is Jane Austen: 0140430721, ...". As shown in this example, the form of the content of the `inform` response should follow the form of the content of the `query-ref` message. To put it simply, the reply answers the question – and no more.

This is not the case with WHOIS++ , our canonical query routing system. When a WHOIS++ client queries a server, it provokes a response giving answers to the query, but may also provoke a response which refers the client to another server. A single query may therefore produce two different types of answer, one of which was not explicitly asked for in the query. To ignore this discrepancy between the

semantics of FIPA queries and WHOIS++ queries and implement a query routing agent system which behaves like WHOIS++ would be to break the semantics of the FIPA ACL.

As a further twist, WHOIS++ does distinguish referral generating queries in one context, that of query expansion. A WHOIS++ client may ask a server for the list of other servers which poll it for centroids (that is, those servers which it knows to have any form of forward knowledge about itself), or for the list of other servers which it polls for centroids (those servers about which it holds any form of forward knowledge). The client uses the list returned to expand the scope of its search, adding the new servers to its agenda just as servers mentioned in referrals would be, but the message it receives from the server is not a referral, strictly speaking. When a server issues a referral, it holds an explicit belief that the referred-to server either can satisfy the query itself, or that it knows of a server which can. When a server answers a `polled-by` or `polled-for` query, the client is making an implicit assumption that the servers contained in the answer are able to satisfy either similar or broader queries to those which may be satisfied by the server itself.

Staying within FIPA, there is a choice to be made between the two main approaches to the representation of query routing protocols. The first solution treats query routing as a new protocol which requires new message types: a type for query messages which expect both an answer and an optional referral in response (call it `query-referral`, and a type for referrals (call it `referral`). This is not unlike the approach taken by KQML for brokering and matchmaking, where a number of special purpose performatives were introduced (`broker(-one,-all)`, `advertise`, `recruit(-one,all)` and `recommend(-one,-all)`). The disadvantage of this approach is that the agent communication language becomes bloated; agent implementors must support an increasing number of rarely-used performatives if their agents are to be able to interact freely with other agents.

The second approach reuses existing performatives and pushes the necessary representation into the content of the message. The FIPA agent communication language takes this approach for matchmaking and brokering; this is performed by sending a `request` message to a suitable agent in order to invoke a matchmaking or brokering service. This approach can be adapted to query routing in two ways. An agent could send a single request containing a compound action, ordered using the sequence operator ; containing the query and a request for referral as shown in Figure 6.2. Alternatively, an agent could send two separate messages, one containing

```
(request
  :sender user
  :receiver agent-1
  :content
    (; (inform-ref
         :sender agent-1
         :receiver user
         :content "query")
       (inform-ref
         :sender agent-1
         :receiver user
         :content "referrals for query")
```

Figure 6.2: Referral query as compound request

```
(query-ref
  :sender user
  :receiver agent-1
  :protocol fipa-query
  :content "query")

(query-ref
  :sender user
  :receiver agent-1
  :protocol fipa-query
  :content "referral query")
```

Figure 6.3: Referral query as separate messages

the query, and one containing the request for referral (expressed as a `query-ref` message).

We have adopted the latter solution for our prototype query routing agent system on the grounds of simplicity. It is more straightforward for the client agent to ask two separate questions ("tell me what satisfies this query" and "tell me who knows about things which satisfy this query", see Figure 6.3) than it is to ask a single compound question, especially since many agents may have been designed to answer only the common FIPA protocols; by splitting the request into two messages, we can make use of the FIPA-Query protocol.

As an aside, a different school of thought views referrals as exceptional occurrences. In agent systems which use a method invocation communication paradigm (instead of the message passing paradigm favoured by FIPA and KQML), a referral may be generated by raising an exception which contains the referral. An example of this, written for the SoFAR agent system (Moreau et al., 2000), is given in Figures 6.4 and 6.5. Figure 6.4 shows the server's implementation of the `query_ref`

```
public Predicate[] query_ref( Predicate p, Envelope e )
  throws MasException {
  if ( p.belongsTo().equals( dirOntology ) ) {
    AgentTerm [] refs = forwardKnowledge.getMatches( p );
    if ( refs != null && refs.length > 0 ) {
      throw new ReferralException( refs );
    } else {
      return kb.getMatches( p );
    }
  }
}
```

Figure 6.4: Exception-based Referrals in SoFAR – server

```
LinkedList agents = new LinkedList();
LinkedList result = new LinkedList();
agents.add( init );
while( agents.size() != 0 ) {
  AgentTerm agent = (AgentTerm) agents.removeFirst();
  try {
    result.addAll( java.util.Arrays.asList(
                     agent.agent().query_ref( q, null ) ) );
  } catch ( ReferralException e ) {
    agents.addAll( e.getReferredAgents() );
  }
}
```

Figure 6.5: Exception-based Referrals in SoFAR – client

method; if the server has forward knowledge relevant to the query p (line 4) it throws an exception containing a list of the agents to which the forward knowledge refers (line 6). Figure 6.5 shows a simple implementation of a client's query routing search algorithm. The client maintains lists of the agents to be asked and the received results (lines 1–2). While there are still agents to be asked (line 4), the client picks an agent (line 5), and asks the query of it, adding any results to the list (line 7–8). If an exception is received (line 9), the list of agents contained in the exception is added to the list of those to be asked (line 10).

Although this implementation of a query routing system works, we believe that it is flawed for two reasons. Firstly, a query routing system must generate referrals as part of its normal operation. Referrals are not exceptional or error cases, and should not be treated as such. Secondly, a method may either return a result or raise an exception, but not both. In query routing systems it is common for a server to return results which satisfy a query and to generate a referral. For these reasons, we believe that agent systems which use method invocation should phrase queries

which expect referrals as two separate queries, one containing the basic query, and one asking which agents can answer the query.

So far, we have dealt only with query routing systems that exclusively use referrals, or to use the agent terminology, those that contain only matchmakers (see Section 3.4). We have ignored those systems which delegate the responsibility for fulfilling queries to those agents which are better placed to answer those queries (that is, a system that uses brokers). Here, the client need only ask the basic query of the servers; it is not interested in information about other agents, and rejects any control that it might have over the processing of the search. Any knowledge that a queried server might have about the abilities of other servers is kept strictly to itself (being used to determine which additional servers it should contact) and is not communicated.

6.3 Representing Forward Knowledge

In the previous section, we have explained our choice of a query routing protocol in which the query and the request for a referral are distinct from each other. We need to be able to represent the forward knowledge which drives the referrals, knowledge about the query answering capabilities of other agents. This knowledge appears both in communications between agents, as they construct the forward knowledge graph and as referrals are issued, and as part of an agent's internal state.

We can characterise the information that an agent holds about specific resources as beliefs. For example, an agent might believe that the entity with the ISBN 0-14-043072-1 is a book entitled "Pride and Prejudice", whose author is named Jane Austen. Forward knowledge, however, is somewhat different. Rather than holding a belief that some other agent believes a certain expression, an agent with forward knowledge believes that the other agent has beliefs *about* some class of expressions.

These generalisations about the beliefs of an agent are an expression of that agent's capabilities, because the beliefs held by an agent affect the communicative acts which it is able to send (according to the semantics of the acts and any social constraints such as truthfulness). This is a different approach to that used in agent communication languages such as KQML. In KQML, the `advertise` and `broker` performatives also contain a representation of an agent's capabilities in their content fields, but this is commonly an opaque expression that contains the name of a service that the agent can provide (for example, the service of answering queries about bibliographic data, or in a rather more limited case, the service of

answering queries about books whose author is Jane Austen). If we were to have taken the approach used by KQML, we would need to introduce a large number of domain ontology-specific terms for representing these services, given the many ways in which an agent could express the generalisation of its beliefs (for example, does it believe 'about books' in general, 'about books' with certain authors, or with certain publishers, or whose titles contain certain phrases).

In the approach that we have taken, we use the defined semantics of our chosen agent communication language (in the first instance, FIPA) to inform our choice of forward knowledge representation. Rather than representing a capability as an opaque expression, or as a simple pattern which matches those query messages which the agent can answer, we represent an agent's capabilities in terms of the beliefs that the agent must hold in order to be able to answer certain types of query, in accordance with the message preconditions specified in the formal semantics of the agent communication language.

In the FIPA agent communications language, the `inform` performative has the precondition that the sending agent believes the content of the message (thus constraining the agent to be truthful). Therefore, when representing the knowledge that an agent is able to send `inform` messages in response to certain `query-ref` messages, we use the beliefs that the agent must hold in order to be able to send those `inform` messages, according to the agent communication language semantics.

Representing the generalised beliefs of an agent presents another choice; we can either attempt to build expressions which denote a range of beliefs using the existing constructs of our knowledge representation language, or we can define a new modality (call it *believes-about*, or `BA`) to represent those beliefs. If we choose the latter approach, we must define inference rules for converting expressions which use this modality of generalised belief into ones which use the more familiar belief modality in order to conform to the semantics of the agent communication language.

This is not a particularly parsimonious solution to the problem; defining a new modality of generalised belief entails extending the knowledge representation language, which will potentially cause interoperability problems with other agents which do not speak the extended knowledge representation language. We choose the former approach, that of using the existing constructs provided by our knowledge representation language, which is more elegant and avoids the Babel-like problem of a proliferation of variant knowledge representation languages. We represent the generalisation of an agent's beliefs using quantification; in (6.1), an agent which believes that $x_1, \ldots, x_n$ are members of a concept c can represent its generalised

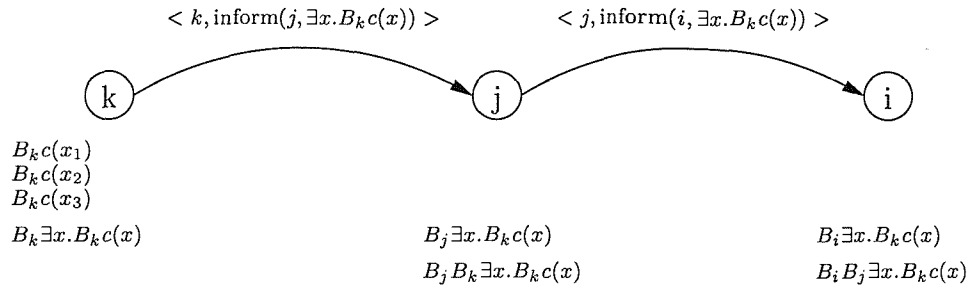


Figure 6.6: Passing FIPA SL encoded forward knowledge

```
(inform :sender k
        :receiver j
        :content (exists ?x (B k (concept ?x))))
```

Figure 6.7: FIPA SL encoded forward knowledge exchange

beliefs by stating that there exists some entity which it believes to be a member of c. For example, an agent which held bibliographic metadata for the books written by Jane Austen could say that there exists some entity such that it believes that entity to be a book written by Austen. The intuition for this approach is that we wish to express the fact that an agent has beliefs about a concept in general, rather than about any specific instance of that concept.

$$B_i c(x_1) \wedge B_i c(x_2) \wedge \cdots \wedge B_i c(x_n) \implies \exists x.B_i c(x) \tag{6.1}$$

The manner in which forward knowledge is passed between agents and so affects their beliefs is illustrated in Figure 6.6. In this diagram, agent k holds a number of beliefs about members of the concept c. It generalises these beliefs as shown in (6.1) to give the expression $B_k \exists x.B_k c(x)$. Note that the expression is cast as another belief (the agent holds beliefs about its own beliefs) – this is necessary in order for the agent to be able to send an `inform` message containing the generalised belief without breaking the semantics of the FIPA ACL.

Agent k sends an `inform` message to agent j (shown in Figure 6.7), which has the rational effect of making j believe the generalisation of k's beliefs ($B_j \exists x.B_k c(x)$). Agent j can also infer k's belief in this generalisation, based on the necessary pre-conditions for the `inform` message ($B_j B_k \exists x.B_k c(x)$).

Agents can now use this forward knowledge to route queries; an example message exchange, after the discussion in Section 6.2, is given in Figure 6.8 (compare with Figure 6.3). In this example, agent i asks agent j which agents hold beliefs about a certain concept, and agent j replies with a list of appropriate agents.

```
(query-ref :sender i
           :receiver j
           :content (iota ?x (exists ?y (B ?x (concept ?y))))
           :protocol fipa-query)

(inform :sender j
        :receiver i
        :content (= (iota ?x (exists ?y (B ?x (concept ?y))))
                    (agent-1
                     agent-2
                     ...))
        :protocol fipa-query)
```

Figure 6.8: FIPA SL referral query

However, the technique we have described for expressing and passing forward knowledge is incomplete; in Figure 6.6, agent i has direct knowledge about the beliefs of agent k, even though this information has been passed through an intermediary. When queried for a referral, agent i will refer the client directly to agent k, rather than following the chain for forward knowledge through agent j. This is at odds with the behaviour of existing query routing systems, where such 'short-circuiting' does not happen.

This discrepancy between the behaviour of our agent-based query routing system and the behaviour of other query routing systems becomes more apparent when the aggregation of forward knowledge is considered. Figure 6.9 shows a system where three agents k_1 to k_3 pass summaries of their beliefs to agent j, which aggregates the summaries and passes the aggregation to agent i. If the forward knowledge were passed as shown in Figure 6.6, agent i's belief database would contain individual direct beliefs for each of agents k_1 to k_3, rather than a single summary belief for agent j which incorporates the belief summaries for agents k_1 to k_3. This will affect our estimates of the routing table sizes (belief database sizes) required for different query routing network topologies given in Chapter 4. In a hierarchical system, for example, the root server would have direct beliefs about every agent in the system, which would give it a routing table size of $|V|$ (where V is the number of agents in the system), and not a constant as we claimed.

Equation 6.2 gives an inference rule that expresses the notion that if an agent believes that another agent has beliefs about a certain concept, effectively it too then has beliefs about that concept. If the beliefs resulting from the application of this rule are passed to other agents in preference to the beliefs which matched the LHS of the rule, the short-circuiting behaviour noted above will be prevented; in

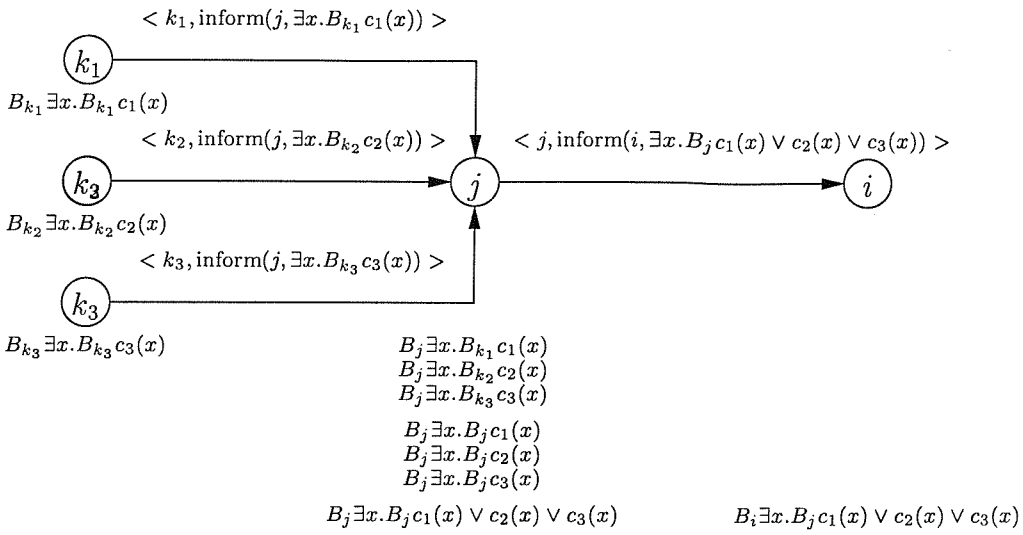


Figure 6.9: Aggregating FIPA SL encoded forward knowledge

the example given in Figure 6.6, agent i will no longer have direct knowledge of k's beliefs, because agent j will have recast its summary of k's beliefs as a summary of its own beliefs.

$$B_i \exists x.B_j c(x) \implies B_i \exists x.B_i c(x) \tag{6.2}$$

Equation 6.3 gives an inference rule which can be used by an agent to aggregate belief summaries. The intuition behind this rule is that if an agent has beliefs about a concept c_1 and a concept c_2, then it has beliefs about the union of these concepts, $c_1 \sqcup c_2$.

$$B_i \exists x.B_i c_1(x) \wedge B_i \exists x.B_i c_2(x) \implies B_i \exists x.B_i c_1(x) \vee c_2(x) \tag{6.3}$$

Figure 6.9 shows the application of both these rules by agent j; the initial beliefs which it formed as a result of the `inform` messages sent by agents k_1 to k_3 are first transformed by (6.2), and then aggregated by (6.3). The aggregated belief summary is then communicated to agent i, so that it holds a single routing belief rather than the three that would have resulted.

These a priori exchanges of information which lead to the formation of forward knowledge or routing beliefs can be classified in terms of which entity initiated the exchange. Either the downstream agent (that which is passing its forward knowledge) initiates the dialogue with the upstream agent (that which receives the forward knowledge and develops the consequent routing beliefs) by sending an `inform` message containing a summary of its beliefs, as in Figure 6.7 (a push model

```
(query-ref :sender i
           :receiver j
           :content (iota ?x (exists ?y (B j ?x))))

(inform :sender j
        :receiver i
        :content (= (iota ?x (exists ?y (B j ?x)))
                    ((concept-1 ?y)
                     (concept-2 ?y)
                     ...))
```

Figure 6.10: Pull model forward knowledge exchange

of forward knowledge distribution), or the upstream agent initiates the exchange by sending an explicit `query-ref` message which asks for a summary of the downstream agent's beliefs, as in Figure 6.10 (a pull model). The choice of push versus pull has an important effect on the latency of the forward knowledge; if the pull model is chosen, there is a tradeoff between the cost of the upstream agent's polls of the downstream agent (the more frequent, the more costly) and the potential obsolescence of the forward knowledge (infrequent polling may lead to increasingly out of date information).

In our Phyle-based query routing system, the message handler which deals with forward knowledge-containing `inform` messages like that in Figure 6.7 applies the rule given in equation (6.2) to the body of the message and adds both the original belief and this new belief to the agent's database. When the agent sends a summary of its beliefs (we have implemented the pull model, so this functionality is located within the message handler for the relevant type of `query-ref` messages), it applies the rule in Equation (6.1) to summarise its own beliefs, and that in Equation (6.3) to integrate this summary with the summaries it has received from other agents.

6.4 Extensionality

In Chapter 5 we described the Phyle agent framework, which makes use of matching techniques such as unification and subsumption for selecting appropriate message handlers. These matching techniques compare structured expressions representing concepts, where the components of these expressions may be thought of as elements of the intensions of the concepts. However, in the model which we presented in Chapter 3, we have defined these matching techniques in an extensional manner such that a concept will unify with another concept if the extensions of the concepts are

non-disjoint (or for subsumption, if the extension of one is a subset of the extension of another).

In doing this, we have made the assumption of *extensionality*, that concepts which have the same extension are the same concept. This is not necessarily the case, because two concepts with the same extension may have different intensions. For example, the extensions of the concepts "the book entitled Pride and Prejudice" and "the book written by Jane Austen, published in 1813" may be the same, but these two concepts are structurally different.

Queries and forward knowledge in a query routing system can both be considered as intensional expressions representing some concept; the process of resolving a query is therefore that of identifying the extension of the query concept. For the query routing technique to be of use in reducing the search scope in a distributed system, the forward knowledge must be designed in such a way that a forward knowledge expression will match a query if the concepts which those expressions represent share some elements of their extensions.

Existing distributed database systems such as Z39.50 (Z39.50 Maintenance Agency, 1995) avoid this issue by adopting common profiles which specify which sorts of query are allowable, one example of which is the Bath Profile (Lunau et al., 2000). These profiles are task-specific interlinguas (specific to the query task) which define the minimum query facilities that a conforming database must support, and enable cross-database searching without expensive query translation (although it should be noted that the query facilities provided by a profile need not be the only query facilities that a database provides - Z39.50 profiles define new interfaces without replacing existing ones). The relation between these profiles and the database schemas to which they are applied is equivalent to the relation between domain-specific ontologies which structure some application domain and the task-specific ontologies which dictate how knowledge acquired in that domain can be manipulated.

In Chapter 3, we described the similarities between queries and forward knowledge, both of which describe some class of resources in an intensional manner. If the use of common profiles can avoid the problem of query translation by restricting queries to use only certain resource properties, a similar approach can be applied to forward knowledge. By defining a common query profile for a query routing system and then restricting forward knowledge expressions to use the same set of resource properties that are permitted in query expressions, we make it more likely that two concepts which have non-disjoint extensions will have intensions that can be unified.

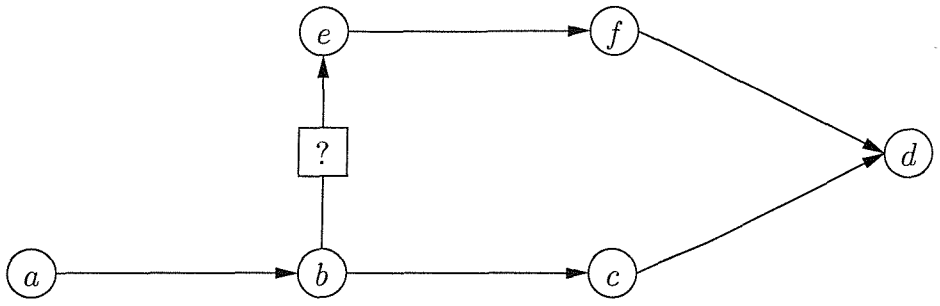


Figure 6.11: Redundant referrals

6.5 Mutual Search Algorithms

In Section 3.4, we introduced the notion of a search with *mutual state*, in which the state of the search is shared between the client and the servers. The existing search systems described in Section 2.2 use only referral (in which the state of the search is held solely by the client) or delegation (in which the state is shared between the servers). Both of these approaches potentially suffer from revisiting previously covered ground, or from following forward knowledge that introduces redundancy.

A system which works by delegation (a brokering system) must coordinate the efforts of those servers to whom the search task has been delegated. If these servers do not communicate with each other (either directly, or via the client) and have no a priori defined search scope, it is likely that they will unwittingly end up querying the same servers more than once, which is wasted effort. A system which works by referral (a matchmaking system) does not suffer from this, because the client alone holds the state of the search, but it does have a different, and more difficult, problem.

Although the client in a referral-based system can avoid querying a server more than once (because it keeps a record of the servers which it has already queried), there is no way for it to determine whether following a chain of forward knowledge will lead to a solution which has either already been found, or would otherwise be found in the future based on the forward knowledge chains that the client already intends to follow.

An example of such a redundant referral is given in Figure 6.11. The first server to be queried by the client is labelled a, and the solution to the query is held by the server labelled d. When queried, a issues a referral for b to the client, this referral being the first in a chain of referrals $a \rightarrow b \rightarrow c \rightarrow d$. When the client follows this referral and queries b, it receives the next referral in the chain ($b \rightarrow c$), and also a referral to e. The referral to e is the first in a different chain $b \rightarrow e \rightarrow f \rightarrow d$

which also leads to d. If the client follows this referral, it is wasting effort because the referral will lead it to a solution which it can reach by a already known route.

The basic algorithm used by referral-based systems (Algorithm 6.1) is an agenda search based on that used in WHOIS++ (Faltstrom et al., 1996). In this, the client maintains an *in-list* (line 1), which contains the servers which the client intends to query, and an *out-list* (line 2), which contains the servers which the client has already queried. The in-list is initialised with the set of servers about which the client has a priori knowledge.

Each iteration, the client selects a server to be queried from the in-list, which has not already been queried (lines 5–6). In line 7, the client queries the selected server for answers which will satisfy q, the query expression, by means of the QUERY() function, any answers being accumulated in the set *Answers*. In line 8, the client requests referrals from the selected server by means of the REFER() function, which are added to the in-list. The selected server is then moved from the in-list to the out-list and another server selected from the in-list. The process terminates when there are no more servers to be selected from the in-list.

Algorithm 6.1: Search for entities which satisfy the query q.

SEARCH(q)

```
1  In ← {servers initially known to this client}
2  Out ← ∅
3  Answers ← ∅
4  while In ≠ ∅ do
5     pick s ∈ In
6     if s ∉ Out then
7        Answers ← Answers ∪ QUERY(s, q)
8        In ← In ∪ REFER(s, q)
9        In ← In \ {s}
10       Out ← Out ∪ {s}
11 return Answers
```

Our definition of the search algorithm uses two additional functions, QUERY() and REFER(), which respectively return answers to a query or referrals to other agents for that query (Equations (6.4) and (6.5)).

$$\text{QUERY}(s, q) = \{x : B_s x \wedge q \sqsupseteq x\} \tag{6.4}$$

$$\textsc{Refer}(s, q) = \{x : B_s \exists y . B_x y \wedge y \sqsupseteq q\} \tag{6.5}$$

This algorithm assumes that the routing tables possessed by each agent contain only belief summaries (to be matched against query terms) and the identity of the agents to which a referral is to be generated if the query matches a summary. In Section 4.4 we discussed the use of traditional network routing algorithms for query routing, and the difficulty of determining whether two routing tables entries will lead to the same destination if potentially non-unique belief summaries are used in place of unique machine addresses.

If the routing tables additionally contain the identity of the agent to whom a forward knowledge chain eventually leads, the server agent has sufficient knowledge to be able to improve this situation. If the client agent presents the server with its in-list and out-list, the server agent is able to prune potentially redundant referrals from the referral set sent to the client agent. These prunable redundant referrals fall into two classes: referrals which will lead to agents that the client has already visited (those in the out-list), and referrals which will lead to agents that the client has already made a commitment to visit in the future (those in the in-list). Note that it is not necessary for the server to prune direct referrals (those in which the agent referred to terminates the forward knowledge chain); the client will do that itself (lines 5–6 in Algorithm 6.1).

These changes to the routing tables effectively require that each server has knowledge of the global topology of the forward knowledge network. Although this is an expensive proposition, and counter to most notions of scalability, query routing systems which use modified network routing algorithms (such as link state - see Section 4.4.4) already flood descriptions of the network topology to all participating servers.

The aggregation of forward knowledge affects the behaviour of such a system which uses such global forward knowledge to prune the referral sets generated by its servers. In Section 3.5.1 we gave a model of the forward knowledge graph which used a labelled digraph to describe the forward knowledge possessed by each server. Figure 6.12 shows a simple network in which two servers, k and l, which respectively contain knowledge that can be summarised by the node labels $\nu(k)$ and $\nu(l)$, pass these summaries to agent j (so creating the edge labels $\lambda(j, k)$ and $\lambda(j, l)$), which aggregates them and passes them to agent i.

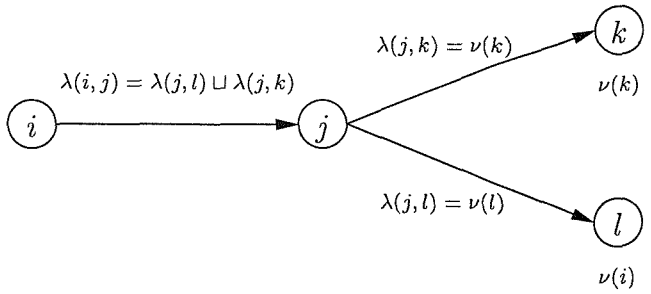


Figure 6.12: Aggregated Forward Knowledge

If the agents' routing tables in this example are augmented as suggested above, this aggregation is problematic. Agent i must have separate beliefs about the contents of k and l (via j) if it is to be able to restrict its referrals based on the client's state, but aggregation of these beliefs into a single belief about j (by the application of the rules in Equations (6.2) and (6.3)) means that i is no longer able to distinguish between the two sources. For forward knowledge which includes the identity of the agent which is its source, aggregation has the effect of splitting forward knowledge chains at the point of aggregation.

This can be partially avoided if we amend the way in which forward knowledge is represented in the system from that described in Section 6.3. The distribution scheme illustrated in Figure 6.6 shows forward knowledge that has not been manipulated by the rule in Equation 6.2, and includes expressions such as $B_i B_j \exists x.B_k c(x)$ which identify both the next and the final agents in a chain of forward knowledge and can be used by a server agent to prune the referral set.

We can express a referral query which specifies the client's in- and out-lists as shown in Figure 6.13. In this example, the atoms `in-list-`x and `out-list-`x are the names of the agents on these lists (note that no distinction is made between those servers which are on the in-list and those on the out-list because the referral set is pruned in the same way for each). This query is structurally similar to that in Figure 6.8 in order that agents which are unable to prune their referrals are still able to provide a response, albeit one which might include redundant referrals.

Therefore, there are two methods of passing forward knowledge as a prelude to a query routing search. The first uses Equation 6.2 to recast the belief summaries in forward knowledge received by an agent as summaries of that agent's beliefs, and is used in systems which aggregate forward knowledge in order to reduce the complexity of distributing the knowledge required by a query routing system. The second retains the identity of the originating agent, and is used in systems which maintain a mutual state between clients and servers for the purpose of eliminating

```
(query-ref :sender i
           :receiver j
           :content (iota ?x (and (exists ?y (B ?x (concept ?y)))
                                  (not (= ?x in-list-1))
                                  (not (= ?x in-list-2))
                                  ...
                                  (not (= ?x out-list-1))
                                  (not (= ?x out-list-2))
                                  ...)))
```

Figure 6.13: Pruned referral query

redundant referrals and reducing the complexity of the search process itself. These techniques complement each other, but are not incompatible. A query routing system can contain both aggregating agents and agents which will prune the referrals that they generate, and we have implemented Phyle agents which do both.

6.6 Domain Ontology Design

In Chapter 1, we introduced the three problem domains which we intend to study, namely bibliographic data, white pages directories and hypertext, these domains being chosen in order to take advantage of readily available datasets. In this section, we describe the design of the agent ontologies used to represent knowledge in these domains.

A central issue faced in all three domains is that of the extent of the designed ontology; there is an important compromise to be struck between the depth and the breadth of an ontology. An ontology may be broad, in that it can describe a wide variety of entities, but also be shallow, in that it describes those entities only in a cursory manner. Conversely, an ontology may be narrowly focussed, and so able only to describe a small set of entities, but be able to describe them in more detail. This is not necessarily an either/or choice. Some ontology designers have chosen both, most notably Lenat (1995) with the Cyc ontology, which is an attempt to create a collection of 'common sense' to be used as a foundation for future AI systems. However, the depth of an ontology is most often curtailed for human, rather than technological, reasons. The majority of the information in these domains is human-authored, and deep ontologies require a greater degree of expertise on the part of the human authors.

Our ontology designs have been informed by the previous work on data representation in the problem domains. We have opted for comparatively shallow ontologies,

matching the available datasets, because we wished to avoid manually annotating the datasets to make them conform to a deeper ontology. We have documented our ontologies following the guidelines suggested by Skuce and Monarch (1990), and these appear in Appendix A; the following sections provide an overview of the design rationale behind the ontologies.

6.6.1 Bibliographic Data

The representation of information about books and journals is a central aim of the library and information science community, and so there are a wide variety of existing solutions, of which we will consider a representative sample. Bibliographic information typically describes the *context* of an information resource, rather than attempting to describe its content. For example, the bibliographic data for a scientific paper might note that the title of the paper is '*An Inquiry into the Causes and Effects of the Variolae Vaccinae*', that it was written by Edward Jenner in 1798 and possibly that it is about smallpox inoculation, but would not describe the full content of the paper itself. For this reason, bibliographic data is also commonly known as *metadata*, because it is data about data.

Perhaps the most widespread bibliographic format is MARC (MARBI/ALA/LOC, 1996), short for *MAchine Readable Cataloguing*, which is used in the majority of library catalogues. MARC records are field-based, with numeric tags denoting some feature of a resource (for example, publisher or subject classification). The value of a field may also be broken down into smaller parts, like the components of an author's name. This structure makes MARC a very rich format, with considerable depth. The role of the bibliographic format here is distinct from, and should not be confused with, the role of the *cataloguing rules* used by a cataloguer. These specify how a record is built from available information, and typically give rules for title capitalisation, name writing and the like. An example of such a set of cataloguing rules are the Anglo-American Cataloguing Rules, 2^{nd} edition (Gorman and Winkler, 1988). MARC is a rich format, and accordingly requires considerable expertise on the part of cataloguers when they create bibliographic records.

There are several MARC formats available, with many countries using their own dialects (USMARC (MARBI/ALA/LOC, 1996), UKMARC (NBS) and CAN/MARC being examples), often for reasons more political than technical. This fragmentation of the MARC format raises both syntactic and semantic concerns for data interchange and translation, and the latter are by far the more pernicious. Purely

syntactic translation occurs when fields from different MARCs share the same meaning but are referred to by different tags. For example, USMARC uses the 020 field for a book's ISBN, while UKMARC uses 021. Semantic translations occur when there is no such mapping, and records in one MARC lose information on translation to another MARC, forfeiting structural detail and becoming simpler in the process.

Over time, there has been a move towards the harmonisation of MARC formats in order to facilitate bibliographic data interchange for services such as union catalogues, but this too suffers from incipient balkanization, with MARC21 (the combination of USMARC and CAN/MARC) and UNIMARC (sponsored by the International Federation of Library Associations) being two different efforts.

The Dublin Core (DCMI, 1999) is a more recent bibliographic metadata format which takes a markedly different approach from that of MARC. Dublin Core (DC) is by design a format with minimal structure, and its development has two main aims. Firstly, it requires less expertise on the part of cataloguers, so Dublin Core records can be authored by those with only minimal training. Secondly, it accepts lossy translations from richer formats as unavoidable. Dublin Core is intended as a format to be used in data fusion (such as the construction of a union catalogue) when the ontologies of the data sources are dissimilar enough to make direct translation between source ontologies impractical.

The Dublin Core metadata schema consists of a set of fifteen fields, designed to be broadly applicable and to have unambiguous, if simple, semantics. A further development, Qualified Dublin Core (Knight and Hamilton, 1997; DCMI), adds qualifiers to some of these fields so that the cataloguer's intent may be better expressed. For example, the `Date` field may be further qualified to indicate whether the given date is the date on which the resource was created, or the date on which it was issued. Qualified DC therefore strikes a compromise between the detail of MARC and the simplicity of unqualified DC.

There has been some interest in one aspect of bibliographic data, subject classification, from within the description logic community (Welty, 1998; Welty and Jenkins, 1999). Subject classification has a long history in the library science community. Today's classification schemes, such as Dewey Decimal or Library of Congress, are firmly rooted in the taxonomic and meronomic trees of knowledge used by libraries five hundred years earlier. Welty's contribution to this has been to clarify the relationship between an information resource (or indeed any entity: persons, events, organizations, etc) and a subject classification. In his ontology, the subject of a resource is modelled as an instance, rather than as a class, thus preserving the

taxonomic relationships between subjects. For example, the subject of a resource may be an instance of the 'medieval history' class. This class is subsumed by the 'history' class, so the subject is also an instance of this latter class (in simple terms, if the book is a medieval history book, it must also be a history book). A similar approach is described in (Pedersen, 1993), which uses relationship lattices for bibliographic information retrieval. These lattices are not unlike description logics (more specifically, the description logic $\mathcal{ALE}$) and allow the representation of concepts such as 'books of fairytales' (or in $\mathcal{ALE}$, `Book` $\sqcap \exists$ `HAS-GENRE.Fairytale`).

Gruber (1994) describes an ontology for bibliographic data as part of a case study of ontology design. His ontology is far simpler than the Dublin Core, but which contains several important features which are also used in this ontology. Gruber defines a unary relation for the references themselves, and defines the other relations as mapping from references to values (authors, titles and so on). In this way, the identity of the bibliographic record is kept separate from the identity of the resource which the record describes. This allows us to write metadata about bibliographic metadata, in order to note the author of a reference for example.

This metadata-about-metadata is referred to by Lassila and Swick (1999) in the RDF specification as higher-order statements. RDF requires the metadata author to explicitly reify the statements which are to be described, although the underlying model treats all statements as reified; the standard RDF syntax hides this for the convenience of metadata authors.

We have chosen to base our ontology on a commonly-used subset of Qualified Dublin Core for pragmatic reasons. It is by far the simplest usable metadata schema in common use and there are well-defined mappings to convert data from existing sources into the Dublin Core. The ontology is illustrated in Figure 6.14 (and described further in Appendix A.1); labels of normal typographical weight are the names of relations (represented in the FIPA SL as binary predicates), emboldened labels are the names of concepts (in FIPA SL, unary predicates) and italicised labels are literal types. This ontology reifies the metadata records itself, which allows us to attach bibliographic provenance to them, and also gives a convenient way of representing records with repeated fields (a paper may have more than one author, and more than one creator field as a consequence).

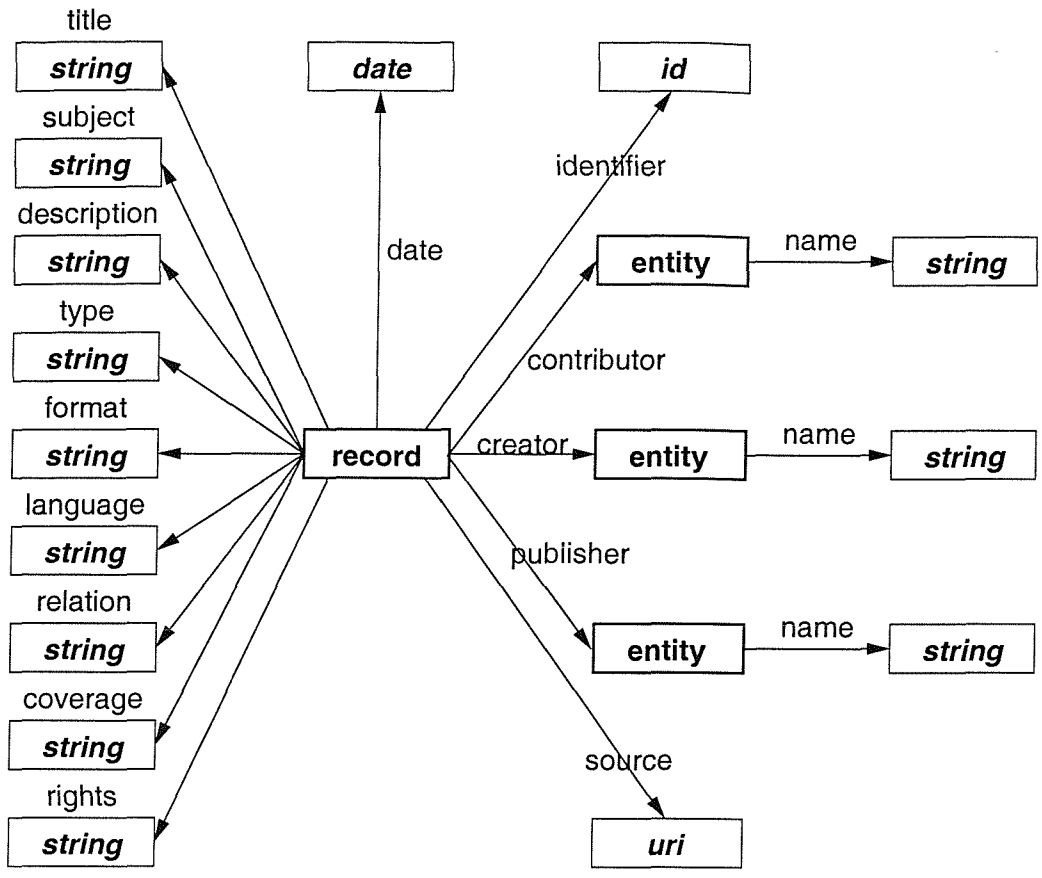


Figure 6.14: The Dublin Core Ontology

6.6.2 Hypermedia

The hypertext research community has constructed a number of formal models of linking over the past two decades, the most well-known of which is the Dexter Hypertext Reference Model, as described by Halasz and Schwartz (1990). Notable features of Dexter are that it separates the structure of an object from its presentation, and treats hypertextual links as a class of objects in their own right, rather than a part of a document. This makes possible the application of different sets of links to a collection of document as variant linking overlays.

In more recent years, the hypertext standardisation effort which begun with Dexter has moved on to the specification of an Open Hypermedia Protocol (OHP), as described by Davis et al. (1998) and Reich et al. (2000).

The OHP specification draws a distinction between different types of hypertext systems, namely *navigational*, *spatial* and *taxonomic* hypertexts, based around a kernel of common services (early versions of OHP received some criticism (Nürnberg and Leggett, 1998; Anderson et al., 1998) for their ignorance of taxonomic or spatial hypermedia). The hypertext systems for which we have built an ontology are

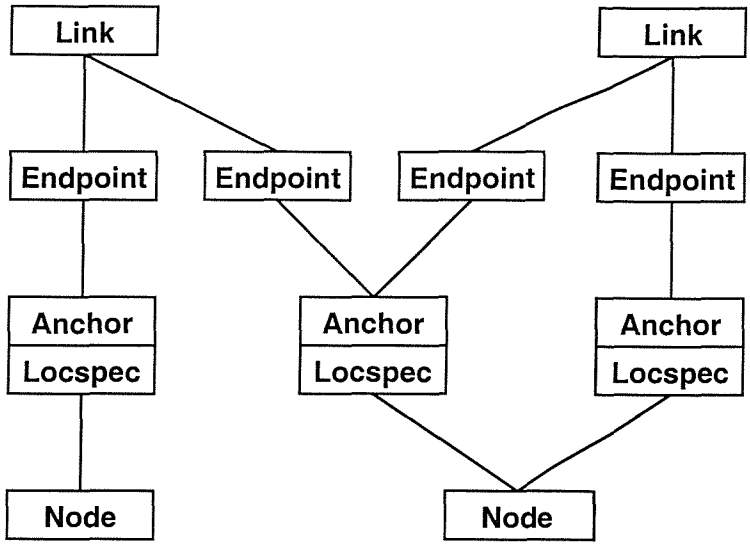


Figure 6.15: OHP Linking Model

primarily navigational, and subscribe to the traditional link-following model of hypertext that uses links as navigable relations which can be followed in order to explore a collection of documents (as opposed to spatial or taxonomic hypermedia which use links primarily to impose structure rather than to provide navigation support), so we have relied on the corresponding navigational subset of the OHP specification (the OHP-Nav specification).

This type of hypermedia is familiar from systems like Microcosm (Fountain et al., 1990), the World Wide Web (Berners-Lee et al., 1994a), and Nelson's Xanadu (Nelson, 1987) (although Xanadu also contains a number of features which are not typically found in navigational hypertext systems). The Fundamental Open Hypertext Model (FOHM) of Millard et al. (2000) unifies these domains by investigating the common structural features of each.

In our design of an ontology for hypermedia, we have concentrated on navigational hypermedia as the most commonly encountered form of hypermedia. Unlike the Dexter model, OHP is specified in terms of the protocol spoken by hypertext servers and clients, which implicitly asserts the underlying data model. The linking model employed by the OHP navigational domain (OHP-Nav) is influenced by that of HyTime (ISO, 1997), in that links, anchors and location specifiers (which identify the location of an anchor in a node, also known as *locspecs*) are treated as distinct entities. The OHP-Nav model also subsumes the model used by the Distributed Link Service (Carr et al., 1995).

An example of the OHP-Nav linking model is shown in Figure 6.15. In this example are two nodes (documents) and two links. One of the links goes from one

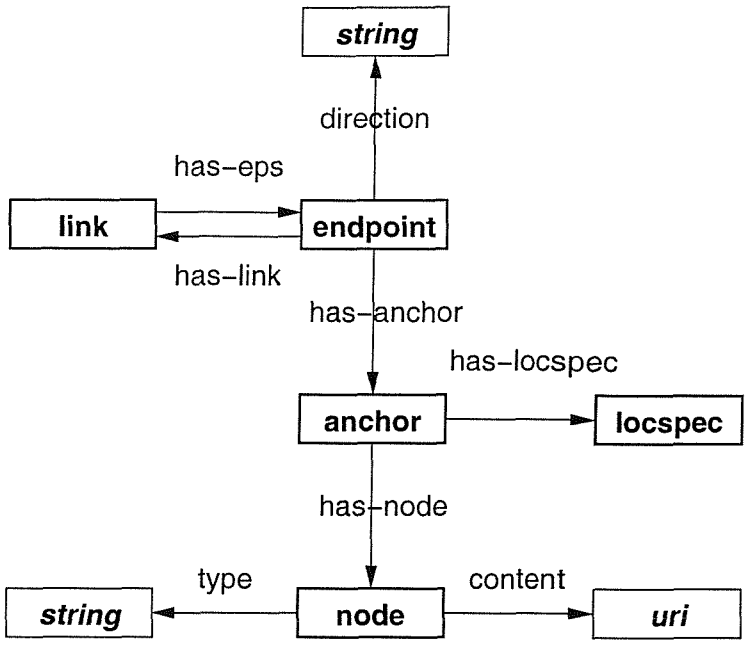


Figure 6.16: The OHP Ontology

node to the other (or rather, from an anchor located in one node to an anchor located in the other), while the other link is between two anchors in the same node.

The existence of the OHP-Nav model means that much of the hard work in designing an ontology - capturing the conceptualisation of the domain - has already been done. An ontology based on this model has been implemented in the FIPA SL (shown in Figure 6.16, and further described in Appendix A.2) and is primarily concerned with the OHP-Nav linking model rather than its other aspects, such as collaborative working.

6.6.3 White Pages

The White Pages domain, which takes its name from the white pages of a telephone directory, involves information about people and organisations, and the ways in which they can be contacted. Our White Pages ontology is based heavily on the schema used by our main datasource, the personnel database of the Department of Electronics and Computer Science (ECSInfo), which is documented in the departmental handbook (ECS, 1999). The ECSInfo schema is strongly similar to that used in the OSI X.500 directory service(ITU, 1993b,c), and also to that used by the vCard electronic business card interchange format (Dawson and Howes, 1998), although this similarity is largely coincidental. Our ontology is illustrated in Figure 6.17 (and further described in Appendix A.3), and allows us to express a set of simple properties that people carry in this domain.

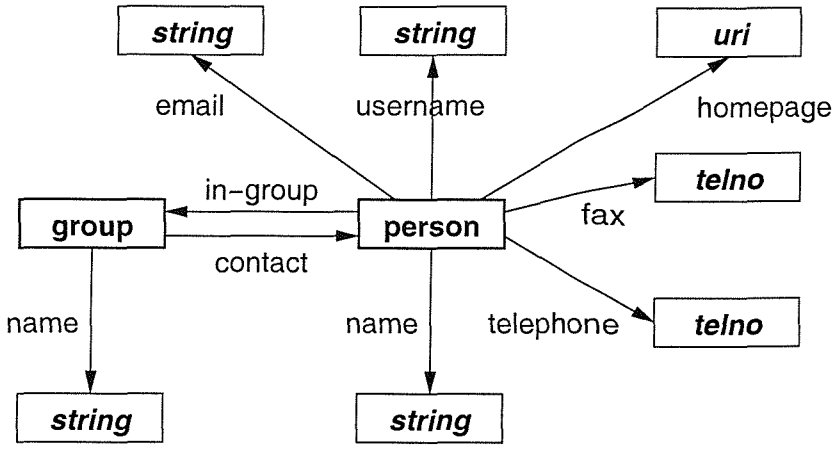


Figure 6.17: The Directory Ontology

6.7 Summary

In this chapter, we have described the key challenges in implementing an agent-based system for query routing search using the Phyle agent framework, including the selection of appropriate communication protocols and knowledge representations for query routing, the design of search algorithms which make best use of forward knowledge by sharing the search state between clients and servers, and the design of application domain-specific ontologies for encoding the data which is to be searched. In the next chapter, we report the results of an empirical study of query routing systems using the network topologies discussed in Chapter 4 which we have used to confirm the conclusions we made regarding the scalability of different query routing systems.

Chapter 7

Experimentation

7.1 Introduction

In Chapter 3 we presented a model of a query routing system which we used to examine the complexity and scalability of a number of different topologies for the network of forward knowledge that underlies query routing systems. In order to increase our confidence in the ability of this model to accurately reflect the behaviour of query routing systems and to confirm our conclusions about scalability, we need to validate the model experimentally.

In order to perform such an empirical validation on a reasonably large scale, we need to build large query routing systems (100,000+ agents). Our agent framework Phyle, which we introduced in Chapter 5, is not capable for this because each agent is too large; a basic Phyle agent requires approximately 2Mb of RAM, and we have insufficient computing resources to be able to run enough agents. The Paraphyle simulator allows us to simulate the behaviour of large query routing agent systems in a lightweight manner by excluding from the simulation all but those components of an agent which are essential to the operation of a query routing system.

In this chapter, we describe the simulation of agent based query routing systems and present the results of an empirical study of the behaviour of these systems which we have used to perform a qualitative validation of our query routing model.

7.2 Approach

Our general approach to simulating query routing systems as described by the model in Chapter 3 is as follows.

We begin by generating graphs of different sizes for the underlying connectivity of a variety of query routing systems, following the discussion on network topology in Sections 4.3 and 4.4. We then label each node in these graphs with an expression which is a summary of the records held by that node. Based on the underlying graph, we then generate forward knowledge for the servers in each of the query routing systems, such that there exists a correctly labelled path (following the discussion in Chapter 4) from the designated entry points of the system (where queries may be presented to the system) to each of the nodes in the system. The number of messages which were required to create the forward knowledge network (the control traffic) is recorded.

Having constructed the forward knowledge networks, we generate queries for presentation to the systems based on the content summaries in the nodes in each system, such that each query can be satisfied by some node in the system (we are not interested in the degenerate case where a query cannot be satisfied even by exhaustively querying each server in the system in turn). We then select the node at which the query will be inserted into the network and begin the processing of the query. The queries are processed in an exhaustive manner in order to identify all the nodes which may satisfy the query, if more than one should exist, and the number of messages which are sent during the processing of the query (the query traffic) is recorded. Because we are searching the system for all the entities which will satisfy the query, the system is effectively a resource or service discovery system rather than a name resolution system; the entities which would be retrieved for a given query are not strictly equivalent, as would be the case with a name resolution system.

The behaviour of a query routing system on processing a query depends both on the topology of the forward knowledge network and on the query itself. In order to reduce any skew introduced into the results due to a particular configuration of the system, multiple instances are generated of each type of query routing system, and multiple queries are processed within each instance.

In this empirical study, we are attempting to confirm our conclusions on the comparative scalability of the query routing systems discussed in Chapter 4, so we generate systems in a range of sizes within a particular class. It was our intention to study the scalability of the different query routing topologies for systems of up to one hundred thousand nodes, but in the case of the more computationally complex systems this was not possible within the limits imposed by the available computing

resources. Where the maximum system size is less than one hundred thousand agents, this has been noted in the initial experiment descriptions in Section 7.4.

7.3 Data Generation

A key consideration in this empirical study is the generation of the data used in the simulation. In our simulation, we do not represent the individual records held by an agent. When simulating a large query routing system, we need to be able to represent a single agent in that system in as lightweight a manner as possible, and individual records are simply too small a level of granularity to be practical for this study. The retrieval from a specified agent of records which match a query is not our concern, but the effort involved in locating that agent is (i.e. we're concerned with database selection, not information retrieval).

In addition, we do not need the intricacies of Phyle's knowledge representation language, but need expressions just detailed enough to be able to determine if one will subsume the other or if they can be unified. We can choose to model expressions either extensionally or intensionally. An extensional model of an expression is a set of the entities that are denoted by the expression, for example, the set of instances in a concept, while the intensional model contains the common characteristics that each member of that concept possesses.

For example, if we take an expression like 'books written by Jane Austen' (or in $\mathcal{ALU}$ Description Logic-like terms, `Book`$\sqcap\exists$`AUTHOR.Jane-Austen`), the extension of the expression contains the books *Sense and Sensibility. Pride and Prejudice*, *Northanger Abbey* and so on, while the common characteristics in the intension are the notions that entities denoted by the expression are books (`Book`), and were written by Jane Austen ($\exists$`AUTHOR.Jane-Austen`). An extensional model of concepts requires that we enumerate all the members of the extension of a concept, which is potentially a large set. Moreover, the more general the concept, the larger its extension.

In Paraphyle, we represent concept expressions intensionally as sets of characteristics which members of the concept must possess. In order to represent the disjunction of concepts (used when aggregating forward knowledge summaries, as in Section 3.5), a given concept is represented by a number of sets of common intensional components, each of which corresponds to a conjunction in the disjunctive normal form rewrite of the concept. Each member of the concept must possess all of the intensional components in at least one of the sets in this disjunction. The

test for the subsumption of a query by a forward knowledge expression or content summary (which correspond to the generation of a referral and the presence of results which satisfy the query respectively) is therefore the subset relation; a query is subsumed by an expression if there is no intensional component of the expression which does not also appear in the query.

We generate content summaries (concept expressions) for each agent by randomly selecting a number of symbols which represent the intensional characteristics which members of the concept must possess. These symbols are selected from a vocabulary containing a fixed number of terms such that all terms in the vocabulary have an equal probability of being selected (i.e. uniform random). It should be noted that, for the purposes of this simulation, we are not concerned with the actual characteristics that these symbols might represent (in the example above, these might be `Book`), and `∃AUTHOR.Jane-Austen`); the symbols are opaque symbols of the form `voc-1`, `voc-2`, `voc-3` and so on.

The content summaries for each agent are therefore represented in Paraphyle by lists of the form `(voc-23 voc-47 voc-92)`, whereas forward knowledge summaries are lists of lists of the form `((voc-44 voc-79 voc-67)(voc-48 voc-25 voc-62))`, where each list within the outer list is the set of intensional components corresponding to a conjunction within the disjunctive normal form rewrite of the forward knowledge concept.

In the following experiments, our content summaries contains three symbols, each of which is drawn with equal probability from a vocabulary of two hundred symbols, unless otherwise specified. An example dataset which was generated for use with Paraphyle is given in Appendix B.

7.4 Experiments

7.4.1 Single Index Server

Query routing systems of this type comprise a single index server which holds forward knowledge for all of the other agents in the system, and are described in Section 4.3.1.

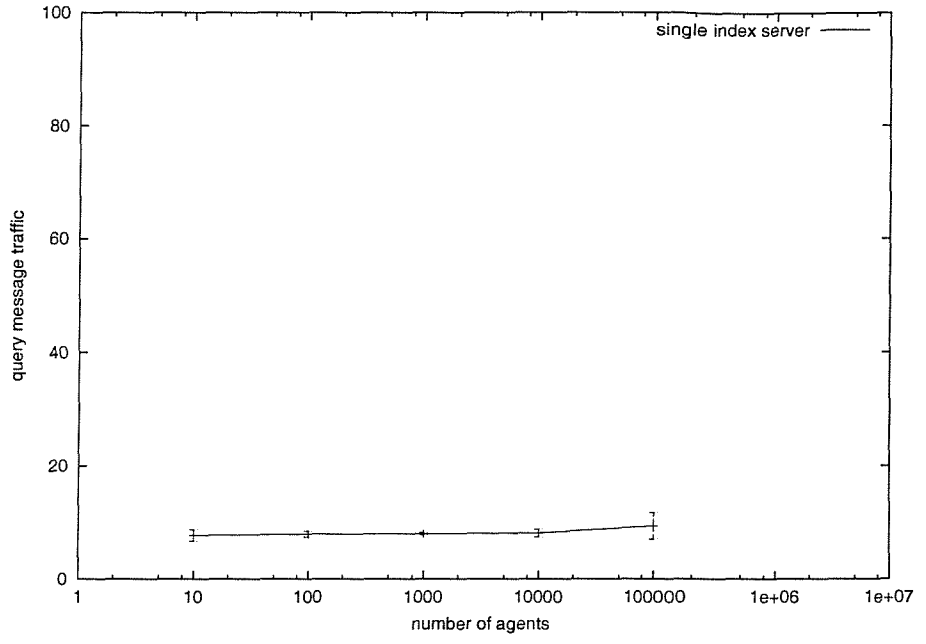
In this experiment, systems were generated containing 10, 100, 1000, 10000 and 100000 agents. For each of these sizes, a total of twenty different systems were generated, and twenty queries generated and processed on each. There is only one entry point for queries in systems of this type, namely the single index server.

Results and Analysis

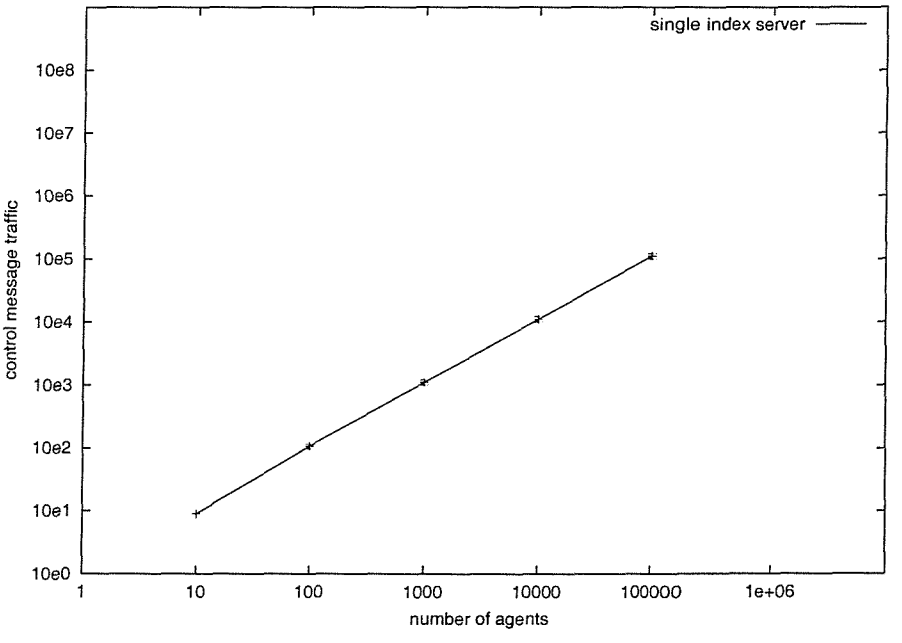
Graphs showing the results for this experiment are given in Figure 7.1. The graph for the query traffic, Figure 7.1(a), shows that the cost of processing a query is almost constant, or increases only very slowly with respect to the number of agents in the system. The graph for the control traffic, Figure 7.1(b), shows that the cost of constructing the forward knowledge network increases linearly with the number of agents in the system.

The slight rise and larger variance for the data point at $n = 100000$ in the query graph is an artifact of the way in which the data was generated. There is a small, but non-zero, probability that given an agent with a particular content summary, another specified agent will have a content summary that subsumes it. In other words, given a query generated from an agent's content summary, there is a slight chance that that will be other agents in the system whose content summaries subsume the query in addition to the agent from which the query was generated.

As the number of agents in the system increases, the probability that there does not exist only a single agent whose content summary subsumes the query also increases; the rise in query messages is due to the existence of multiple agents which can satisfy a given query.



(a) Query traffic



(b) Control traffic

Figure 7.1: Results for single index server

7.4.2 Hierarchy

Query routing systems of this type form a strict hierarchy in which agents at higher levels have forward knowledge about agents at lower levels (not necessarily direct forward knowledge), and are described in more detail in Section 4.3.3.

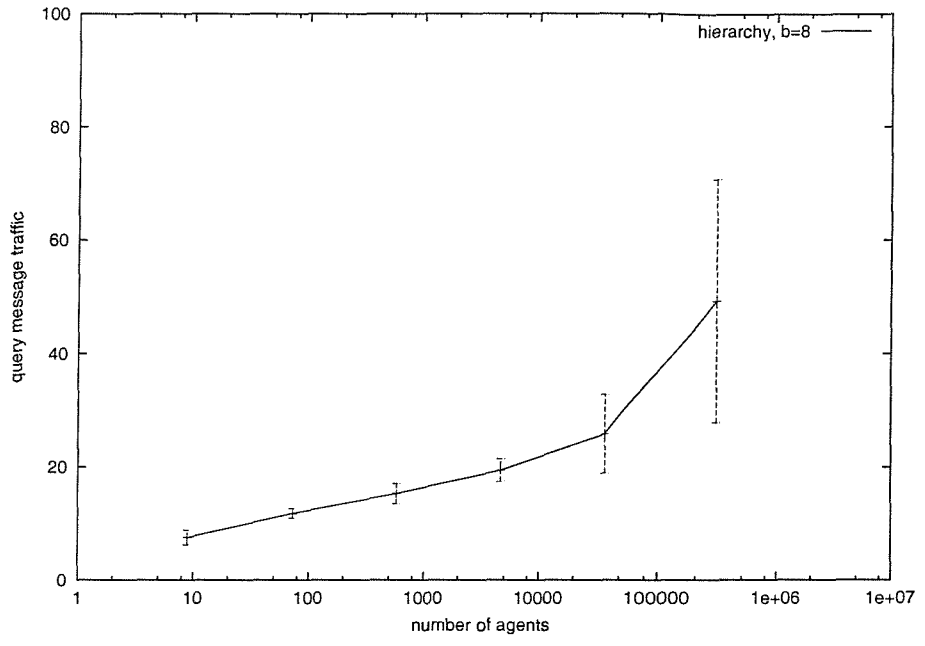
In this experiment, systems were generated such that each non-leaf agent has eight children (breadth $b = 8$) for depths ranging from 2 to 7 (i.e. from 9 to 299593 agents, calculated as $(b^d - 1)/(b - 1)$). As in Section 7.4.1, a total of twenty different systems were generated for each of these sizes, and twenty queries generated and processed on each. All queries were inserted into the system via the root node.

Results and Analysis

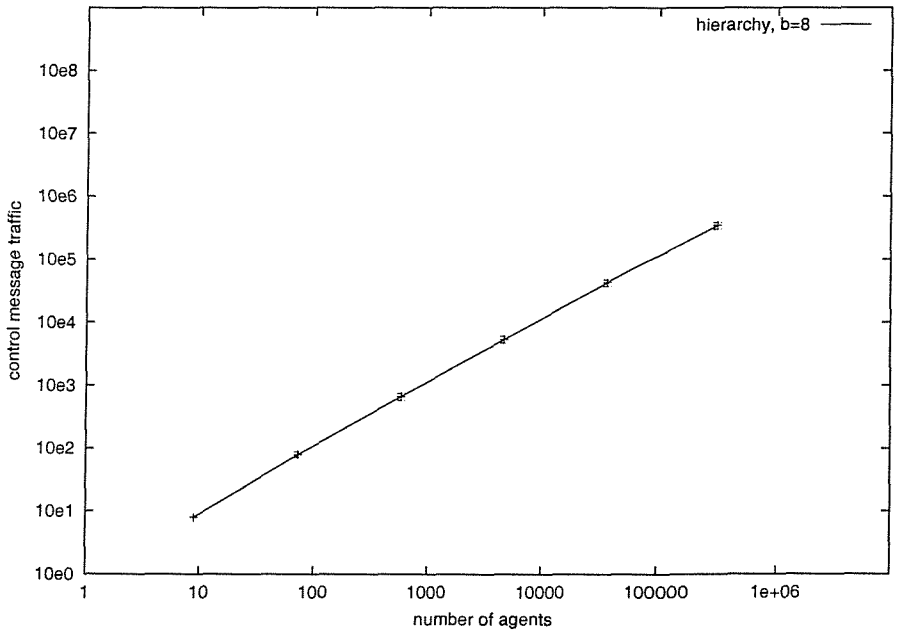
The results for this experiment are given in graphical form in Figure 7.2. The graph for the query traffic, Figure 7.2(a), shows that the cost of processing a query increases logarithmically with increasing system size, while the graph for the control traffic, Figure 7.2(b), shows that the cost of constructing the forward knowledge network increases linearly with system size.

The data generation artifact which caused the rise in later data points in the previous experiment is also present in this experiment. The magnitude of the rise is greater than before because the agents which can satisfy a query are now further from the point of entry than they were in the previous experiment, where the index server was one hope away from all of the data servers. This causes a more visible rise in the query traffic because a longer chain of forward knowledge entails the sending of a larger number of query messages in order to traverse that path, so magnifying the artifact.

The effect that vocabulary size has on the occurrence of multiple solutions is illustrated in Figure 7.3. In this experiment, the size of the vocabulary (the number of terms from which expressions are constructed) was increased by a factor of ten, from 200 to 2000 (as explained in Section 7.3, the members of the vocabulary are opaque symbols). In this figure, there is no rise in the query traffic for larger systems because the larger vocabulary has reduced the probability of duplicate answers existing in the system.

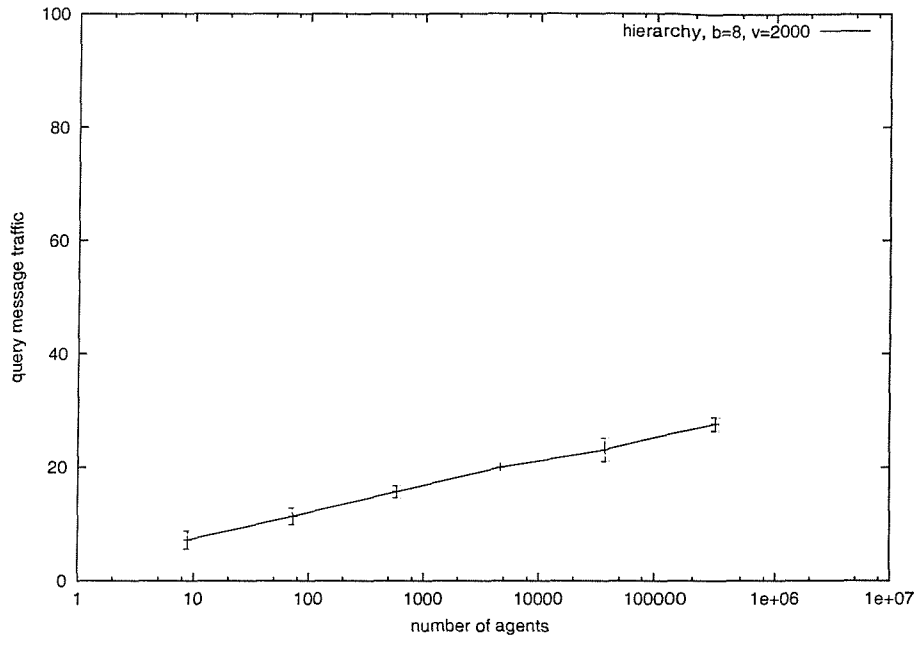


(a) Query traffic

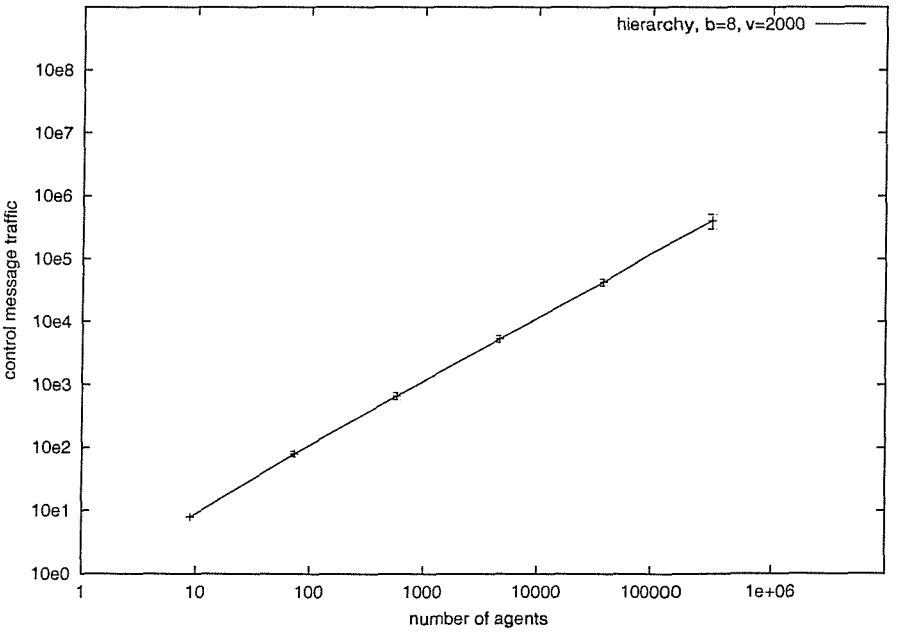


(b) Control traffic

Figure 7.2: Results for hierarchy



(a) Query traffic



(b) Control traffic

Figure 7.3: Results for hierarchy with large vocabulary

7.4.3 Hierarchy with Search Expansion

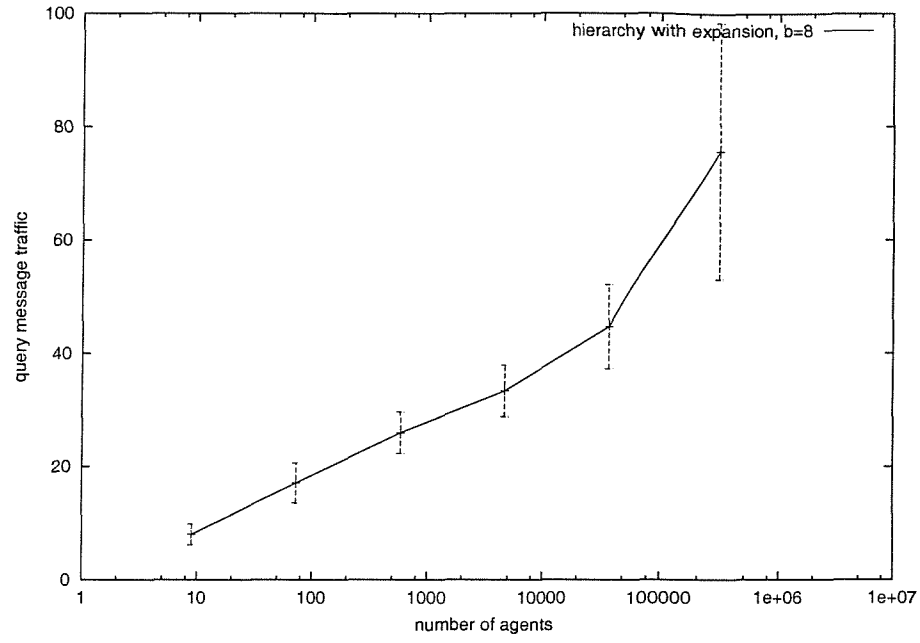
Query routing systems of this type are like the hierarchical systems of the previous experiment, but include forward knowledge that leads from the leafs of the tree towards the root, so that queries can be inserted anywhere in the systems, not just at the root node.

In this experiment, systems were generated such that each non-leaf agent has eight children (breadth $b = 8$) for depths, d, ranging from 2 to 7 (i.e. from 9 to 299593 agents). As in Section 7.4.1, a total of twenty different systems were generated for each of these sizes, and twenty queries generated and processed on each. An insertion point for each query was chosen at random from the nodes in the system.

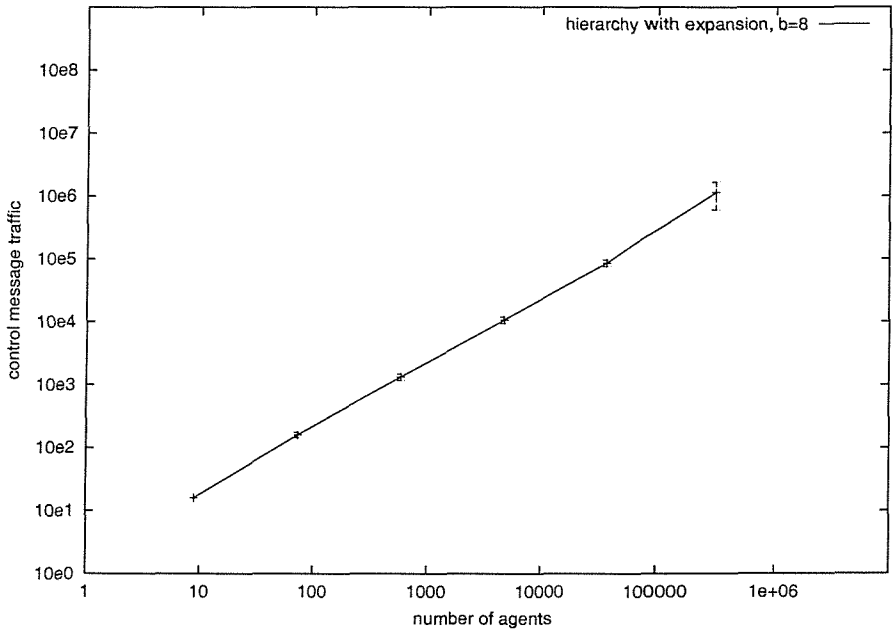
Results and Analysis

The results for this experiment are given in Figure 7.4. The control traffic for this experiment, Figure 7.4(b), shows that the cost of constructing the forward knowledge network increases linearly with system size, and is increased by a constant factor from that for a simple hierarchy.

The query traffic, shown in Figure 7.4(a), shows that the cost of processing a query increases logarithmically with increasing system size (with the same proviso as before with regards to the artifact introduced by the vocabulary size). However, because queries may be introduced into the system at any node rather than just at the root, the maximum path length from the insertion point to the goal is now $2d - 2$, rather than $d - 1$ as it was in a simple hierarchy, which accounts for the greater query traffic generated by this system in comparison to the simple hierarchy in Section 7.4.2.



(a) Query traffic



(b) Control traffic

Figure 7.4: Results for hierarchy with search expansion

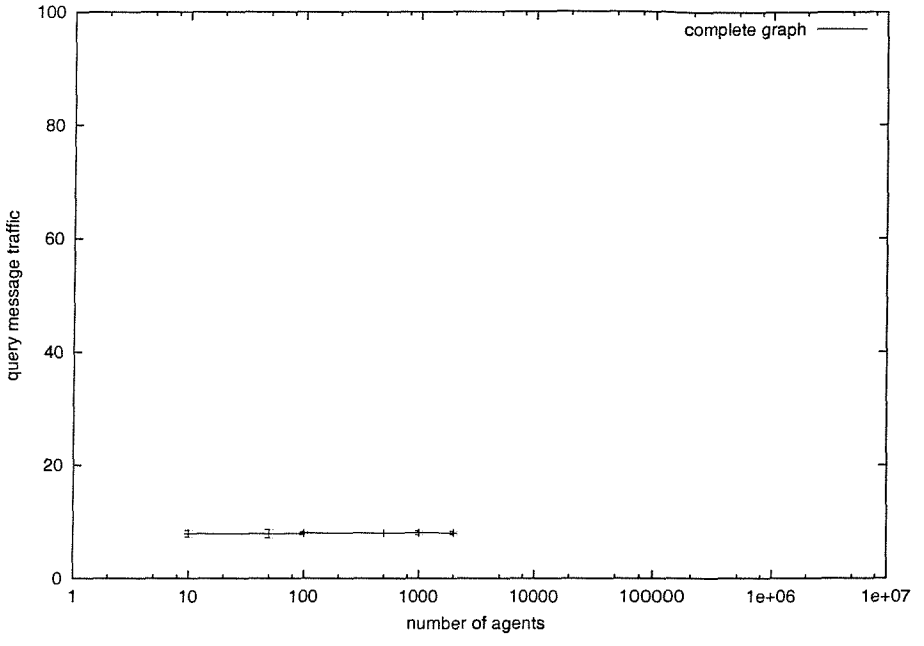
7.4.4 Complete Graph

In systems of this type, every agent has direct forward knowledge about every other agent, as described in Section 4.3.6. Queries may therefore be inserted at any point in the system.

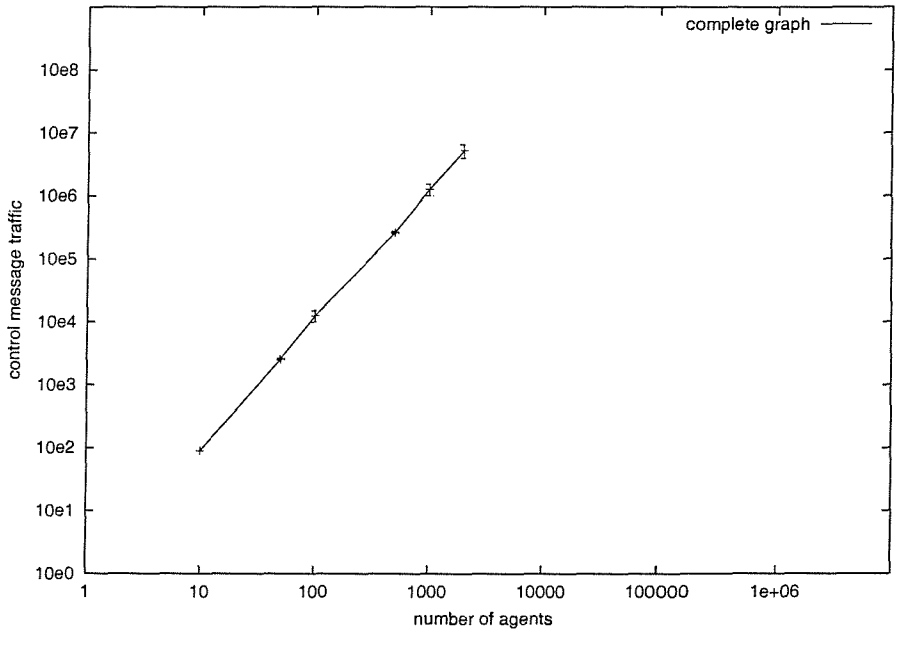
In this experiment, systems were generated with 10, 50, 100, 500, 1000 and 2000 agents. Note that the largest system studied here contains fewer than 100000 agents due to constraints imposed by the available computing resources. As in earlier experiments, a total of twenty different systems were generated for each of these sizes, and twenty queries generated and processed on each. An insertion point for each query was chosen at random from the nodes in the system.

Results and Analysis

The results for this experiment are given in graphical form in Figure 7.5. The query traffic for this experiment, Figure 7.5(a), shows that the cost of processing a query is constant with increasing system size. However, the control traffic, Figure 7.5(b), indicates that the cost of constructing the forward knowledge network increases polynomially as the square of system size, making this an expensive system to build.



(a) Query traffic



(b) Control traffic

Figure 7.5: Results for complete graph

7.4.5 Councils

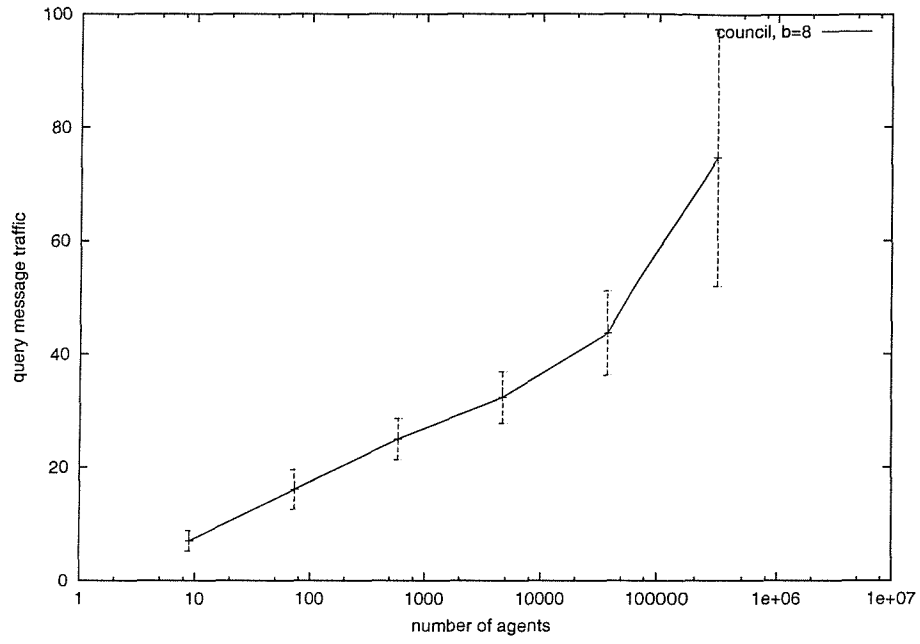
Council networks are a cross between hierarchical networks and complete graphs in which the nodes are arranged into layers with each layer being fully connected internally (described previously in Section 4.3.7).

In this experiment, the systems that were generated contained councils of size 9, in order that the results be comparable with those obtained for hierarchical systems (a given council within the network corresponds to a node and its direct children in a hierarchical network, hence a breadth of 8 for a hierarchical network requires a council size of 9 for the corresponding council network). We generated systems containing between 1 and 6 layers of councils, again to facilitate comparison with the equivalent hierarchical network. As in earlier experiments, a total of twenty different systems were generated for each of these sizes, and twenty queries generated and processed on each. An insertion point for each query was chosen at random from the nodes in the system.

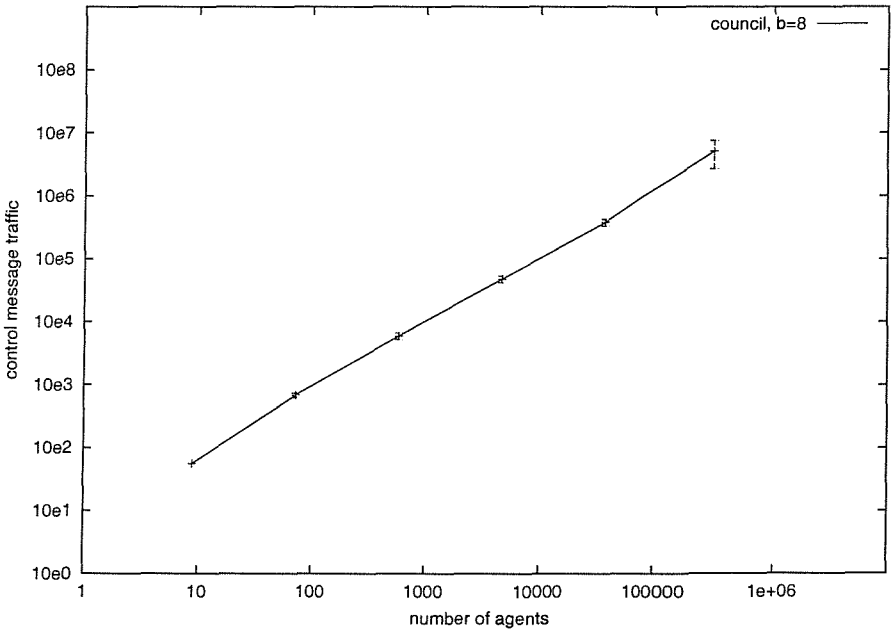
Results and Analysis

Graphs showing the results for this experiment are given in Figure 7.6. The query traffic for this type of system increases logarithmically with increasing system size (shown in Figure 7.6(a)) though at a slightly lower level than did the traffic for the hierarchical system with search expansion in Section 7.4.3 (a comparison of the query traffic for these two systems is given in Figure 7.7). The difference between the two traffic levels is constant, and is due to the way in which this type of network tries to improve on the efficiency of query routing search. A council network effectively contains a shortcut which can be used when a path passes through two siblings; instead of having to route the query via their common parent, the query may be passed directly from one to the other. However, the processing of a query in a council network involves at most one query transfer of this type, which lies at the apex of the query's path from source to goal, so the advantage offered by this type of network is limited to the elimination of a single referral.

The control traffic for these networks, Figure 7.6(b), shows that the cost of constructing the forward knowledge network increases linearly with increasing system size, but also that this cost is a constant factor higher than the equivalent cost for hierarchical systems. This constant factor arises from the exchange of messages which is necessary to make each inidividual council a clique within the network.



(a) Query traffic



(b) Control traffic

Figure 7.6: Results for council

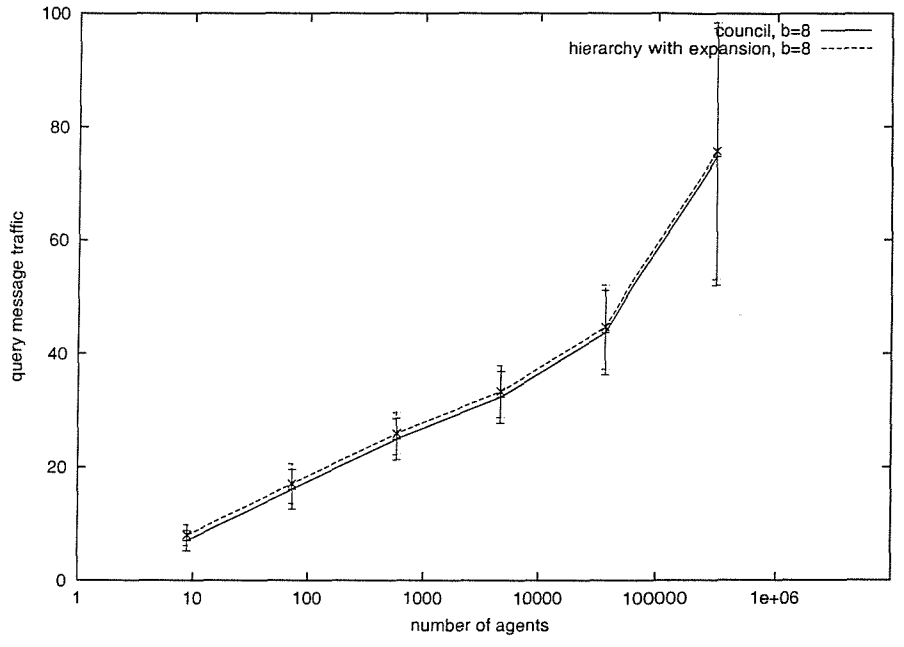


Figure 7.7: Comparison between council and hierarchy with search expansion

7.4.6 Flooding

Flooding from a single source is used to propagate queries in Gnutella-like systems. In our simulation of these systems, we represented the underlying network topology by constructing a disordered graph following the methods outlined for small world networks (Watts, 1999) and for scale-free networks (Barabási and Albert, 1999). The construction of these networks takes different parameters; for the small world network, we started with the ring lattice $L_{n,4}$ and rewired with probability 0.25, and for the scale free network we started with the complete graph K_4 and grew the graph by adding $n - 4$ nodes, each with degree 4.

In this experiment, systems were generated with 10, 50, 100, 500, 1000 and 2000 agents (again due to available computing resources). As in previous experiments, a total of twenty different systems were generated for each of these sizes (and for each network type), and twenty queries generated and processed on each. An insertion point for each query was chosen at random from the nodes in the system. When an agent first receives the query, it sends on it to all of its neighbours. Subsequent receipts of the query by an agent are ignored (i.e. do not result in the query being passed to neighbouring agents).

No measurements were made for control traffic in this experiment (Gnutella does not construct a forward knowledge network).

Results and Analysis

A graph showing the results for this experiment is given in Figure 7.8. The query traffic for this system increases linearly with increasing system size, which is consistent with the prediction in Chapter 4 of $O(|E|)$ complexity for the flooding operation.

Small world and scale free networks are sparse graphs ($|E| \ll |V|^2$) where the number of edges are proportional to the number of vertices (due to the fixed neighbourhood size of the unmodified ring lattice for small world networks, and the fixed degree for new vertices in scale free networks), so a flooding operation which requires that the message be sent once only over every edge in the network would be expected to scale linearly. It should be noted, however, that this is a considerably more expensive operation than the logarithmic or constant complexity solutions considered earlier in this chapter.

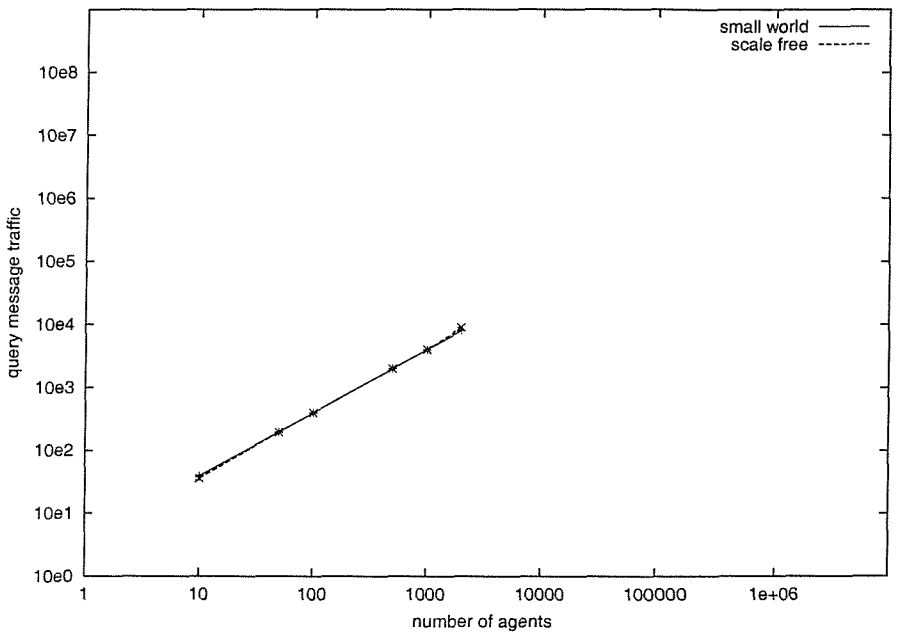


Figure 7.8: Results for flooding

7.4.7 Distance Vector

As in Section 7.4.6, we represented the underlying network topology by constructing disordered graphs following the methods outlined for small world and scale free networks and using the same parameters (for the small world network, we started with the ring lattice $L_{n,4}$ and rewired with probability 0.25, and for the scale free network we started with the complete graph K_4 and grew the graph by adding $n-4$ nodes, each with degree 4).

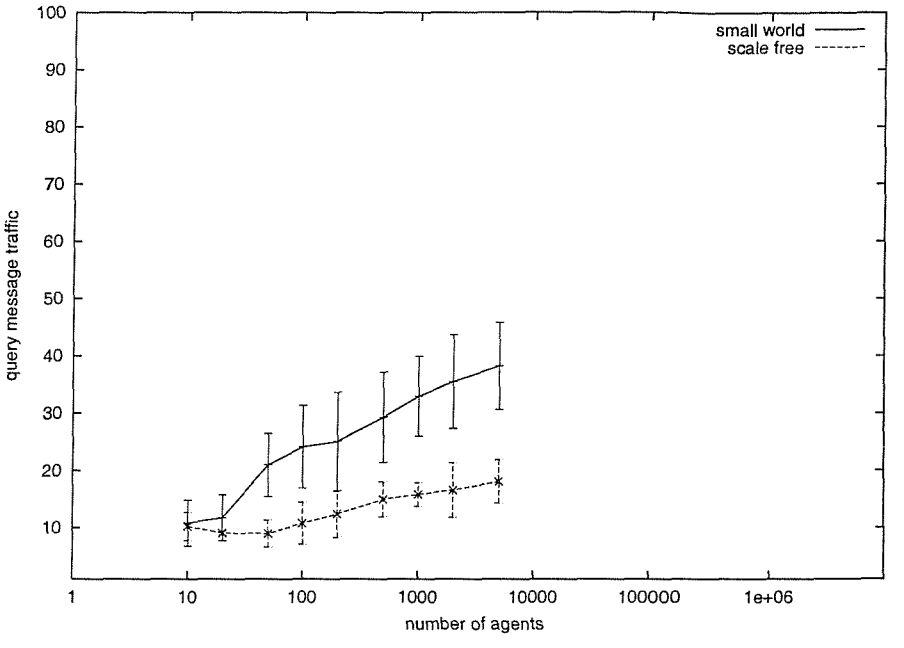
In this experiment, systems were generated with 10, 20, 50, 100, 200, 500, 1000, 2000 and 5000 agents. As in previous experiments, a total of twenty different systems were generated for each of these sizes (and for each network type), and twenty queries generated and processed on each. An insertion point for each query was chosen at random from the nodes in the system.

The forward knowledge network was constructed by running the distance vector algorithm (altered slightly to use the augmented routing tables discussed in Section 4.4), halting when the routing tables in the systems had converged (i.e. when a round of the algorithm produced no change in the routing tables).

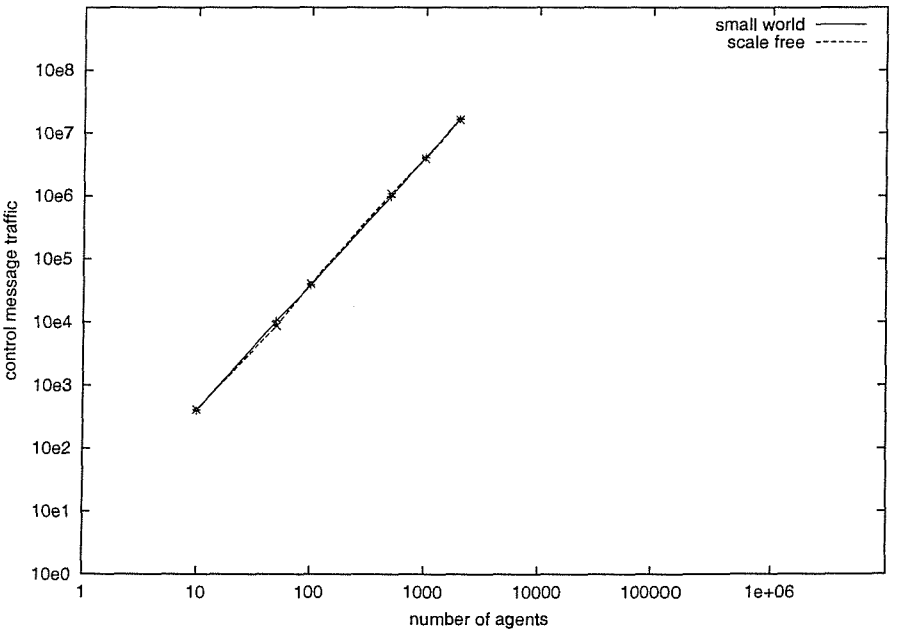
Results and Analysis

Graphs showing the results for this experiment are given in Figure 7.9. The query traffic for this type of system increases logarithmically with increasing system size (Figure 7.9(a)) on both small world and scale free networks, which agrees with the predictions made in Chapter 4.2. However, the query message traffic for scale free networks increases more slowly than that for small world networks, which suggests that for the rewiring probability we have used, small world networks have a larger diameter than scale free networks with equal number of vertices and edges.

The control message traffic for both systems increases at the same rate as $|V|^2$ (see comparison with the control message traffic for complete graphs in Figure 7.11), which is consistent with our prediction in Section 4.4 of $O(|V||E|)$ because both scale free and small world networks have $|E|$ proportional to $|V|$ (in both cases, adding a node to the networks adds a fixed number of edges).



(a) Query traffic



(b) Control traffic

Figure 7.9: Results for distance vector

7.4.8 Link State

As with the experiments for flooding and distance vector routing, we represented the underlying network topology by constructing disordered graphs following the methods outlined for small world and scale free networks and using the same parameters (for the small world network, we started with the ring lattice $L_{n,4}$ and rewired with probability 0.25, and for the scale free network we started with the complete graph K_4 and grew the graph by adding $n-4$ nodes, each with degree 4).

In this experiment, systems were generated with 10, 20, 50, 100, 200, 500, 1000, 2000 and 5000 agents. As in previous experiments, a total of twenty different systems were generated for each of these sizes (and for each network type), and twenty queries generated and processed on each. An insertion point for each query was chosen at random from the nodes in the system.

The construction of the network of forward knowledge was simulated by flooding each agent's local topology to all other agents and then running our modified version of Dijkstra's algorithm (Algorithm 4.1). The manner in which this was performed differed from that given in Section 7.4.6; each tick, every agent sends to its neighbours all the content expressions that it received for the first time in the previous tick, so combining several pieces of forward knowledge into a single message. In order to reduce the effort involved in running this simulation, Dijkstra's algorithm was not run for every agent in the system, but rather only for those agents identified as insertion points for queries; these agents therefore have complete knowledge of the topology of the network (and in particular, its shortest paths), which is used to simulate the processing of a query by the system.

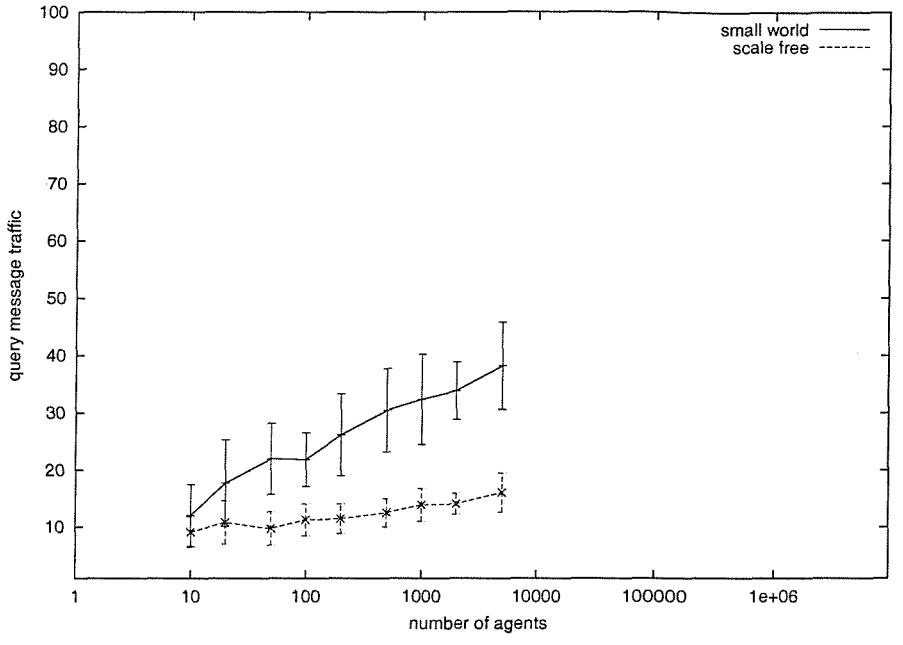
Results and Analysis

Graphs showing the results for this experiment are given in Figure 7.10. The query traffic for this type of system increases logarithmically with increasing system size (Figure 7.10(a)) on both small world and scale free networks, which agrees with the predictions made in Chapter 4.2.

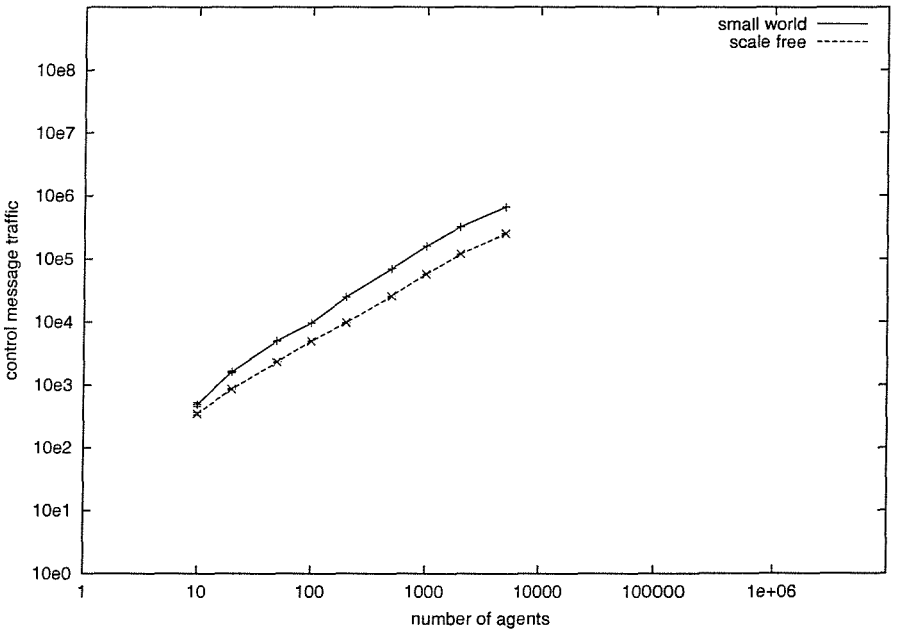
As with the distance vector experiment, query message traffic for scale free networks increases more slowly than that for small world networks, which suggests that for the rewiring probability we have used, small world networks have a larger diameter than scale free networks with equal number of vertices and edges. The control message traffic for both systems appears to increase linearly with increasing

number of vertices, which is at odds with our earlier prediction of $O(|E| \log |V|)$ complexity.

However, as mentioned in Section 7.4.7, $|E|$ is proportional to $|V|$ for scale free and small world networks (so the complexity could be restated as $O(|V| \log |V|)$), and over the range of system sizes studied, the logarithmic factor in this complexity introduced by the graph diameter has a much smaller effect on the measured traffic than does the number of edges.



(a) Query traffic



(b) Control traffic

Figure 7.10: Results for link state

7.5 Discussion

In the previous section, we presented the results of the experiments which we have used to examine the scalability of different network topologies for query routing systems. The systems which we have studied can be divided into three rough categories: those which try to minimise the cost of querying the system without regard to the cost of building the forward knowledge network, those which minimise the cost of building the network without regard to the cost of processing queries, and those which attempt to trade off the cost of querying and network building against each other.

An example of the first category is the complete network, which has a constant querying cost but requires that every agent should contact every other agent, and an example of the second category is the Gnutella-like flooding system, which does not use forward knowledge, but which requires that a query be sent to every agent. The remaining systems are all characterised by a query cost which scales logarithmically as the system grows in size, but they approach the problem in two significantly different ways. One set of systems constructs an ordered forward knowledge network which consists of an explicit hierarchy of agents, while the other set assumes that the underlying network topology will be disordered and attempts to construct the forward knowledge network using adapted network routing algorithms.

The routing algorithms used in these disordered systems do not scale as well as the ordered systems; in Chapter 4 we gave control message complexities from $O(|E| \log |V|)$ to $O(|E||V|)$ for these systems. The disordered networks (small world and scale free) that we used to simulate these systems are sparse, and have $|E|$ proportional to $|V|$ (in both cases, adding a node to the network adds a fixed number of edges), so in effect the complexities range from $O(|V|^2)$ to $O(|V| \log |V|)$.

The graph in Figure 7.11 contrasts the control message traffic for distance vector and link state routing with that for complete graphs and hierarchies. As can be seen in this, the control message traffic for distance vector grows at the same rate as for complete graphs (verifying our prediction of $O(|V|^2)$ complexity), while the traffic for link state routing (the all pairs flooding operation) grows at a considerably slower rate (it appears to be linear in this graph because $\log |V| \ll |V|$).

Our approach to studying the scalability of query routing systems by considering the complexity of query processing and the construction of the forward knowledge network makes the assumption that the cost of performing these tasks is adequately represented by the communication complexity, but this fails to take into account

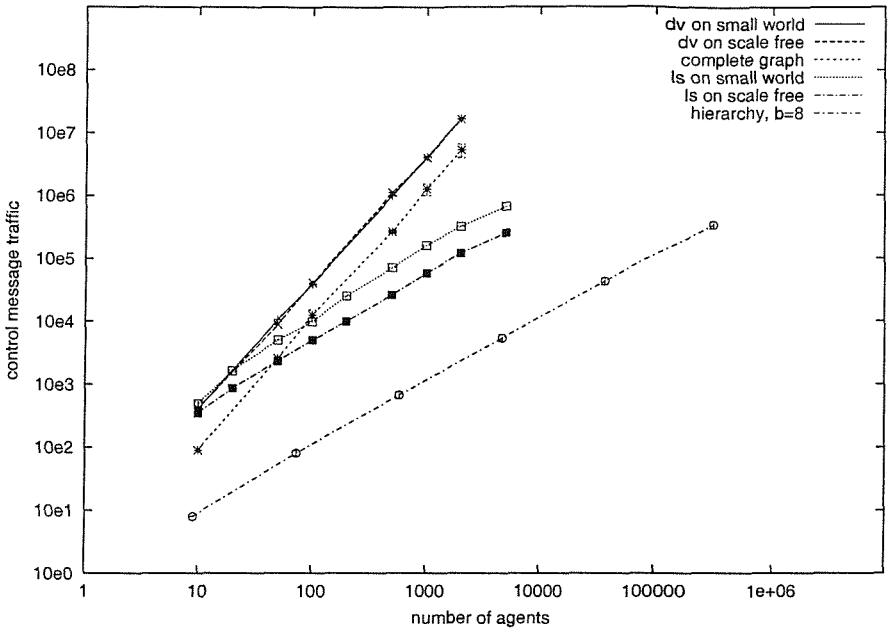


Figure 7.11: Control complexity comparison

other costs of a social or political nature which are associated with the creation and maintenance of an ordered system (the cost of agreeing on the placement of agents in the hierarchy, for example — contrast this with the low organisational costs of more anarchic peer-to-peer systems such as Gnutella or Freenet). The maintenance of hierarchical systems like DNS requires a great deal of human effort that perhaps should be accounted for, if only it were possible to quantify it; unfortunately, these costs are largely implicit. Our overall recommendations for scalable query routing systems must therefore take a number of factors into consideration:

- The frequency of queries is greater than the frequency of updates, so it is more important that the cost of processing a query be low.
- After the cost of processing queries, it is important to reduce the cost of constructing and maintaining the network of forward knowledge.
- Ordered systems may involve hidden organisational costs which cannot be easily quantified.

The first of these concerns forces us to rule out flooding based systems such as Gnutella, while the second excludes the complete graph and distance vector systems. Of the remaining systems, the single index server places too great a load on a single component, leading to a potential bottleneck and single point of failure.

The choice which remains is between the hierarchical systems (including councils) and the disordered systems which use link state routing (see Table 4.1). Given our characterisation of the underlying disordered network topology as a small world

or scale free network, both feature logarithmic query complexities. The control complexity for the two are different, with hierarchies remaining linear as opposed to the linear-log complexity of link state systems, but the logarithmic factor grows slowly enough to have little effect over the range of system sizes we have studied. Finally, if the expansion of query scope is allowed in hierarchical systems, the cost of propagating an update scales linearly with the size of the system, as it also does for the link state systems ($|E|$ being proportional to $|V|$ in small world and scale free networks). This leaves the hidden organisational costs of ordered systems as the only remaining criterion; in the absence of any other criteria, our recommendation is therefore for query routing systems which do not assume that the system is ordered, and which use our modified link state algorithm to construct the forward knowledge network.

7.6 Summary

In this chapter, we have described a series of experiments which we have conducted using the Paraphyle simulator (described in Section 5.3) in order to validate our model of a query routing system (described in Chapter 3), and to confirm our conclusions with respect to the effects on scalability of different forward knowledge network topologies (see Chapter 4).

In Chapter 8, we will summarise the contributions we have made in this work, and discuss potential avenues for further development of the agent systems we have built and future research topics which expand on the work presented in this thesis.

Chapter 8

Further Work and Conclusion

8.1 Conclusion

In this thesis, we have presented a model of the query routing paradigm which we have subsequently used to inform the design of scalable agent systems for query routing. The main objective of this work was to study the behaviour, and in particular the scalability, of query routing systems, an area which has received little attention to date. To achieve this objective, we performed an empirical study of a variety of network topologies for query routing systems which we used to confirm the conclusions which we had drawn on the complexity of constructing and using systems based on these topologies, these conclusions being based on our study of the model we created.

The concept of query routing is a pattern which occurs frequently in distributed systems which need to perform the resource discovery or name resolution tasks, as was discussed in our survey of existing systems in Chapter 2. The majority of the systems studied were hierarchical in nature, but the swiftly growing popularity of peer-to-peer computing (Napster, Gnutella and so on) suggests that a less restrictive approach to the construction of the network of forward knowledge that underpins a query routing system can promote wider participation in such a system.

In order to study query routing systems in the abstract, we built a graph theoretic model of the query routing process, described in Chapter 3, with which we were able to characterise a number of different system topologies, based on the systems discussed in the previous chapter. This is an important contribution of this work, since no such model of the query routing paradigm existed previously.

In Chapter 4, we settled on the use of a family of partially disordered graphs to represent the anarchic, but not entirely structure-free, coalitions of users that

characterise many peer-to-peer systems. We suggested that where most traditional query routing systems construct their forward knowledge networks in a hierarchical manner, these more loosely organised systems might benefit from the application of network routing algorithms which have been suitably modified for use in the query routing domain.

Using our query routing model, we determined the scalability of the different system topologies under examination in terms of the communication and space complexity of processing queries and of constructing the network of forward knowledge which is necessary for correct operation of a query routing system. We believe that this particular section of this thesis is potentially of great use to designers and implementors of peer-to-peer systems, because it provides an objective comparison of the performance and scalability of peer-to-peer systems based on different underlying topologies.

We then turned our attention to the application of the query routing technique to the problem of coordination in multi-agent systems (in particular to the service discovery task). In Chapter 5, we discussed the design of our agent framework Phyle, with particular emphasis on its novel features, such as fluid message handlers. Although the Phyle agent framework was not explicitly designed for the construction of query routing agent systems, there were no suitable alternatives available at the time we carried out this stage of our work; had such alternatives been available, we would have used them in preference to the task of reinventing the wheel. Our views on the future development of the Phyle agent framework are summarised in Section 8.2.1.

While it was possible to build small multi-agent systems containing a handful of agents in Phyle, the framework was not lightweight enough to permit the construction of large scale systems containing many thousands of agents. In order to address this, we designed the Paraphyle simulator, which used a minimal representation of an agent which was specifically tuned to the requirements of query routing.

In Chapter 6, we discussed the issues that affect the design of query routing agent systems, with particular emphasis on the semantics of queries and referrals. This work made use of our model in order to explain the way in which agents in a query routing agent system aggregate forward knowledge during the construction of the forward knowledge network. We feel that this chapter is a particularly novel contribution to the development of mediator systems for multi-agent systems.

Finally, in Chapter 7 we used the Paraphyle simulator to study the behaviour of large scale query routing agent systems in order to confirm the conclusions on the

scalability of different query routing system topologies that we had drawn following our analysis in Chapter 4, and in doing so validate our model.

Query routing is a complex domain, and there are a number of aspects which we did not study in this work. In particular, the effects of failures and other aspects of unreliable systems on the effectiveness of database selection in query routing systems of various topologies is still an open issue which we feel merits future attention. In Section 8.2.2, we have identified two possible avenues for future research and summarised our immediate intuitions as to potential techniques which could prove useful in these areas.

8.2 Future Work

During the course of this work, a number of areas of interest came to our attention which we were not able to further develop or study due to time constraints. In this section, we summarise the areas which we consider to be worthy of future research and outline a possible path for the future development of the software discussed in this thesis.

8.2.1 Future System Development

There are two areas in which future systems development must be considered, namely further development of the Phyle agent framework (see Chapter 5) and the development of agent-based query routing systems (see Chapter 6), not necessarily using the Phyle framework.

The Phyle agent framework was born out of necessity; when the implementation phase of this study was conducted, there was no suitably mature and freely available agent framework. This is no longer the case, and there are a number of well rounded agent frameworks which conform to standards, such as those published by FIPA (of which FIPA-OS (Poslad et al., 2000), Jade (Bellifemine et al., 1999) and Zeus (Nwana et al., 1999) are good examples). Although Phyle has some features which are not present in other frameworks (such as its fluid message handlers), it also lacks some features which are present in those frameworks (particularly integration with system facilities or other external services like LDAP).

Conversely, the implementation of query routing systems such as those described in this thesis is not inextricably tied to the Phyle framework. Indeed, the best way to encourage a more widespread use of query routing-like systems for service

discovery in multiagent systems would be to implement them in more widely used agent frameworks.

The increasing complexity of emerging agent standards like FIPA places a greater workload on the implementors of agent frameworks. If compliance with these standards is to be maintained, this work is best spread amongst a community following the open source model (Raymond, 1999). For this reason, we believe that the Phyle agent framework should not be further developed, and that any future system development should take the form of query routing implementations for other agent frameworks.

8.2.2 Future Research Directions

There are two areas related to query routing which we believe merit further investigation in the long term. The first is the robustness of the system, or its ability to cope with failures. The second is the effect of unfaithful or incomplete forward knowledge on the effort required to process queries.

Robustness and Failure

An area which has not been addressed in this work is the robustness of a query routing system, that is the effects of failure on its behaviour. A promising approach to a study of this area uses work from statistical physics, namely *percolation theory* (Grimmett, 1999). This is a branch of stochastic graph theory which deals with the effects of varying the interconnections in a random system, and is used in applications from studies of the spread of disease in a population to the porosity of concrete.

The basic idea of percolation is that the overall connectivity of a system varies with the probability that a given connection exists in such a way that there is a sharp transition (a phase transition) from a disconnected to a connected graph when the probability rises above some critical value (the percolation threshold). This transition is not unlike the transition to a small-world network noted by Watts and Strogatz (1998); in (Watts, 1999), Watts comments on the similarity between these effects.

The topologies studied in Chapter 4 treated the connectivity of the network as an invariant property; in the given networks, the techniques used for propagating forward knowledge were such that there would exist an appropriately labelled path from the network entry points to every other node in the network. The existence of

these paths means that if there is an answer to a query somewhere in the system, it should be possible to locate that answer using the query routing technique (giving the query a perfect recall).

There are two types of failure which may occur in a query routing system. In the first instance, a server may fail (known as *site percolation*). If this server holds the answer to the query, the query will obviously fail, as will also be the case if there is a single path to the location of the answer and the failing server lies on that path. Secondly, one of the edges in the network may be removed (known as *bond percolation*) as a server loses a piece of forward knowledge, which again affects the processing of a query if that edge is part of a path which leads to the answer to the query.

Percolation theory is of interest to us because it allows us to determine which topologies are more robust in the event of failure. If the property which we are interested in maintaining under failure is the existence of correctly labelled paths to all destinations, the higher the percolation threshold for this property, the more robust the system.

Robustness clearly has some relation with redundancy (of paths or forward knowledge, and not necessarily of the referrals generated from that forward knowledge). Part of a future study of the robustness of query routing systems should include an investigation of the effects of redundancy on robustness, and the conditions under which redundancy best improves robustness (i.e. where to add redundant forward knowledge to best improve robustness).

Unfaithful Forward Knowledge

In Section 3.3.2 we introduced the forward knowledge effectiveness measures, *faithfulness* and *completeness*, which are used to express how well an expression summarises a set of records.

In our investigation of the effects of different topologies on the behaviour of query routing systems, we have made the assumption that the forward knowledge in the system is both faithful and complete. This is not necessarily the case (and in fact is frequently the opposite in real world systems), but it does give us an upper bound on the effectiveness of a query routing system.

There are two main sources of inaccuracy in a query routing system's forward knowledge. It may either be inaccurate from the outset due to the poor summarisation of a server's records, or it may have been formed by the lossy aggregation of forward knowledge from other sources.

The former source was outside the remit of this study because we were primarily interested in the database selection problem. While poor summarisation affects the effectiveness of the use of query routing for database selection, it is not something that can be mitigated against within the realm of conventional query routing systems. If a database advertises its capabilities incorrectly, a query routing system cannot detect this when it is building its routing tables without exhaustively querying such a database to determine whether the summary is correct.

Lossy aggregation typically occurs because a server wishes to reduce the size of the intension of a forward knowledge summary (in order to reduce the storage or transmission requirements). The intension size is reduced by removing selected characteristics from the summary expression, but this has the effect of increasing the extension of the summary so that it denotes more records than it should; the summary thus becomes unfaithful.

Regardless of its source, the effect of unfaithful forward knowledge is to introduce false positives in the process of database selection. While these do not necessarily affect the overall effectiveness of a query routing system (no records which match the query are hidden from the querying client), they do represent an increase in the effort required to obtain results from the system because the false positives are not identified as such until they are queried.

A possible solution to these problems, and a fruitful area for future research, is the introduction of adaptive behaviour into query routing systems, so that queries which fail to produce predicted results are used to modify the forward knowledge on which the prediction of results was made. These modifications could be incremental and take the form of counter examples induced by failed queries (e.g. this server knows about Victorian novels, except those written by Charles Dickens), or they could be full updates in which a server which requests a new forward knowledge summary to replace one which it believes to be incorrect. In this way, adaptive forward knowledge could also be used as an alternative to the explicit updating of forward knowledge discussed in Section 3.3.5.

The issue of which entity controls a query routing search reappears here, in that a server which issues referrals is unaware of the success or failure of the client to which it provided the referral unless that client informs it of its status (so behaving as a 'good neighbour', which recalls Singh's work (Singh, 1998) on agent communication languages which are defined in terms of social norms). Conversely, a server to which has been delegated the task of processing a query is relying on its own forward knowledge, and so is aware when that forward knowledge gives false positives.

Relying on failure alone to trigger forward knowledge updates requires that there are sufficient queries to exercise the forward knowledge. Forward knowledge which relates to seldom queried servers would potentially have greater latency and could stagnate. In the worst case, a domain which combined infrequent queries with rapidly changing data would result

A future study of the effects of unfaithful forward knowledge should therefore include an investigation into the comparative complexities of explicit forward knowledge updates and adaptive forward knowledge, and the time complexity of convergence in systems with adaptive forward knowledge.

Appendices

Appendix A

Domain Ontologies

The ontologies in this appendix were designed and documented following the guidelines suggested by Skuce and Monarch (1990) and Uschold and King (1995).

A.1 Bibliographic Metadata

Our ontology for bibliographic metadata (described in Section 6.6.1) is based on the terminology introduced by the Dublin Core effort (DCMI, 1999), and is centred around the concept of a record which bears metadata as a proxy for the resource which is being described (as opposed to a resource-centric view where the bibliographic data is attached directly to the resource).

Name	Type	Description
title	Relation	A name given to the resource described by this record.
subject	Relation	The topic of the content of the resource.
description	Relation	An account of the content of the resource.
type	Relation	The nature or genre of the content of the resource.
format	Relation	The physical or digital manifestation of the resource (signified by a MIME media type).
language	Relation	A language of the intellectual content of the resource.
relation	Relation	A reference to a related resource.
coverage	Relation	The extent or scope of the content of the resource.
rights	Relation	Information about the rights held in or over the resource.
date	Relation	A date associated with an event in the life cycle of the resource (typically the creation of the resource).
identifier	Relation	An unambiguous reference to the resource described by this record within a given context (a URI).
contributor	Relation	An entity responsible for making contributions to the content of the resource.
creator	Relation	An entity primarily responsible for making the content of the resource.
publisher	Relation	An entity responsible for making the resource available.
source	Relation	A reference to a source from which this resource is derived.

Table A.1: Bibliographic Ontology

A.2 Hypermedia

Our hypermedia ontology (introduced in Section 6.6.2) is based on the data model used by the Open Hypermedia Protocol (Reich et al., 2000).

Name	Type	Description
link	Class	An entity which provides an association (possibly navigable) between nodes.
endpoint	Class	The endpoint (source and/or destination) of a link.
anchor	Class	An entity which denote a node or a location within a node (denoted by a locspec) which can be used as the endpoint of a link.
locspec	Class	A location specifier which denotes a particular point or range in a node.
node	Class	An entity which may be linked.
direction	Relation	Indicates the direction of a particular endpoint: source, destination or bi-directional.
has-endpoint	Relation	Indicates that a link contains a particular endpoint. The inverse of the has-link relation.
has-link	Relation	Indicates that an endpoint is used as part of a particular link. The inverse of the has-endpoint relation.
has-anchor	Relation	Indicates the anchor with with a particular endpoint is associated.
has-node	Relation	Indicates the node to which an anchor refers,
type	Relation	Indicates the media type of a node: image, text, audio, etc.
has-locspec	Relation	Indicates the location within a node to which an anchor refers.
content	Relation	Indicates the location of the content of a node (i.e. the address of a file or resource)

Table A.2: Hypermedia Ontology

A.3 White Pages

Our White Pages ontology (described in Section 6.6.3) is based on a simplified version of the class model of the X.500 directory (ITU, 1993b,c) and on the schema used by the Department of Electronics and Computer Science (ECS, 1999).

Name	Type	Description
person	Class	A person.
group	Class	A named group of people or other organisation (department, research group, etc).
email	Relation	The email address of a person.
in-group	Relation	Indicates memberhop of a person in a group.
contact	Relation	Indicates the person who is the contact name for a group.
username	Relation	Indicates the username of a person.
name	Relation	The name of a group (required).
fax	Relation	The fax number for a person.
telephone	Relation	The telephone number for a person.
homepage	Relation	The URI of the homepage of a person or group.
personal-name	Relation	The personal (first) name of a person.
family-name	Relation	The last (family) name of a person.
title	Relation	The title (Mr, Dr, Rev, etc) of a person.

Table A.3: White Pages Ontology

Appendix B

Sample Simulation Data

This appendix contains a sample dataset which was generated for use with the Paraphyle simulator. This dataset describes a network of agents (labelled `hier-1` to `hier-21`) which form a hierarchy of three levels in which each agent has four direct children (see diagram in Figure B.2).

The network is represented as a list of agents; each agent in the list is represented as a list as shown in Figure B.1. The first item in the list (line 1) is a symbol used as the agent name, in this case `hier-15`. The second line is the content summary for the agent, which in this case is the expression `voc-33` ⊓ `voc-98` ⊓ `voc-118`. The remaining lines give the forward knowledge held by the agent, represented as a list of pairs where the first part of each pair is the forward knowledge summary expression and the second part is the name of the agent about which this forward knowledge is held. For example, the agent `hier-15` has forward knowledge about agent `hier-12` such that it believes that `hier-12` knows about expressions which are subsumed by `voc-121` ⊓ `voc-114` ⊓ `voc-145`.

```
1   (hier-15
2    (voc-33 voc-98 voc-118)
3    ((((voc-176 voc-66 voc-78)) . hier-11)
4     (((voc-121 voc-114 voc-145)) . hier-12)
5     (((voc-40 voc-9 voc-101)) . hier-13)
6     (((voc-64 voc-115 voc-101)) . hier-14)))
```

Figure B.1: Sample dataset for a Paraphyle agent

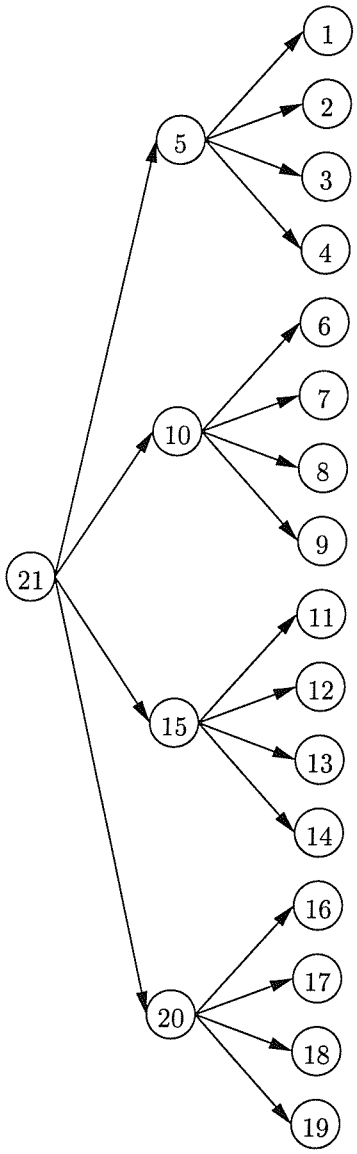


Figure B.2: Graph for sample dataset

```
((hier-21
  (voc-73 voc-62 voc-19)
  ((((voc-17 voc-90 voc-49)
     (voc-75 voc-3 voc-32)
     (voc-153 voc-83 voc-147)
     (voc-196 voc-75 voc-143)
     (voc-101 voc-93 voc-191)) . hier-5)
   (((voc-99 voc-192 voc-138)
     (voc-1 voc-164 voc-117)
     (voc-178 voc-114 voc-94)
     (voc-77 voc-84 voc-188)
     (voc-125 voc-5 voc-13)) . hier-10)
   (((voc-33 voc-98 voc-118)
     (voc-176 voc-66 voc-78)
     (voc-121 voc-114 voc-145)
     (voc-40 voc-9 voc-101)
     (voc-64 voc-115 voc-101)) . hier-15)
   (((voc-193 voc-92 voc-143)
     (voc-114 voc-186 voc-10)
     (voc-67 voc-37 voc-132)
     (voc-143 voc-94 voc-126)
     (voc-120 voc-168)) . hier-20)))
 (hier-20
  (voc-193 voc-92 voc-143)
  ((((voc-114 voc-186 voc-10)) . hier-16)
   (((voc-67 voc-37 voc-132)) . hier-17)
   (((voc-143 voc-94 voc-126)) . hier-18)
   (((voc-120 voc-168)) . hier-19)))
 (hier-19
  (voc-120 voc-168)
  ())
 (hier-18
  (voc-143 voc-94 voc-126)
  ())
 (hier-17
  (voc-67 voc-37 voc-132)
  ())
 (hier-16
  (voc-114 voc-186 voc-10)
  ())
 (hier-15
  (voc-33 voc-98 voc-118)
  ((((voc-176 voc-66 voc-78)) . hier-11)
   (((voc-121 voc-114 voc-145)) . hier-12)
   (((voc-40 voc-9 voc-101)) . hier-13)
   (((voc-64 voc-115 voc-101)) . hier-14)))
 (hier-14
  (voc-64 voc-115 voc-101)
```

```
 ())
(hier-13
 (voc-40 voc-9 voc-101)
 ())
(hier-12
 (voc-121 voc-114 voc-145)
 ())
(hier-11
 (voc-176 voc-66 voc-78)
 ())
(hier-10
 (voc-99 voc-192 voc-138)
 ((((voc-1 voc-164 voc-117)) . hier-6)
  (((voc-178 voc-114 voc-94)) . hier-7)
  (((voc-77 voc-84 voc-188)) . hier-8)
  (((voc-125 voc-5 voc-13)) . hier-9)))
(hier-9
 (voc-125 voc-5 voc-13)
 ())
(hier-8
 (voc-77 voc-84 voc-188)
 ())
(hier-7
 (voc-178 voc-114 voc-94)
 ())
(hier-6
 (voc-1 voc-164 voc-117)
 ())
(hier-5
 (voc-17 voc-90 voc-49)
 ((((voc-75 voc-3 voc-32)) . hier-1)
  (((voc-153 voc-83 voc-147)) . hier-2)
  (((voc-196 voc-75 voc-143)) . hier-3)
  (((voc-101 voc-93 voc-191)) . hier-4)))
(hier-4
 (voc-101 voc-93 voc-191)
 ())
(hier-3
 (voc-196 voc-75 voc-143)
 ())
(hier-2
 (voc-153 voc-83 voc-147)
 ())
(hier-1
 (voc-75 voc-3 voc-32)
 ()))
```

Glossary

ACL *Agent Communication Language* The language used to express the intent of the messages that an agent sends.

Cluster A group of related data objects.

Centroid A type of summary object which summarises a number of data objects, most commonly those which form a cluster.

Cluster representative A summary of a cluster.

Content summary A summary of the data held by an entity in a distributed information system.

Data access The task of retrieving a particular data object from a distributed information system.

Database selection The task of identifying the entity within a distributed database which can satisfy a particular query.

FIPA *Foundation for Intelligent Physical Agents* Standards body which has defined a standard ACL (the FIPA ACL) and a knowledge representation language (sl, or semantic language).

Forward knowledge Information held by an index server which contains a content summary for another server, and which is used for query routing.

Index server A server in a distributed information system which holds forward knowledge about other servers.

KIF *Knowledge Interchange Format* A knowledge representation language defined by the Knowledge Sharing Effort.

KQML *Knowledge Query Manipulation Language* An ACL defined by the Knowledge Sharing Effort.

Knowledge Sharing Effort DARPA-funded US agent standardisation effort which produced KQML and KIF.

Query routing A technique for distributed search which uses forward knowledge to constrain the scope of the search.

Resource discovery The task of searching a distributed information system for data objects with particular characteristics

Service discovery The task of searching a distributed information system for entities which can provide a particular service.

Bibliography

H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. Knight, R. Nagpal, E. Rauch, G. Sussman, and R. Weiss. Amorphous computing. AI Memo 1665, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1999.

AgentBuilder. AgentBuilder: An integrated toolkit for constructing intelligent software agents. White paper, Reticular Systems, Inc., Jan. 1998. Available online at `http://www.agentbuilder.com/`.

R. Albert, H. Jeong, and A.-L. Barabási. The diameter of the world wide web. *Nature*, 401:130–131, 1999.

R. Albert, H. Jeong, and A.-L. Barabási. Error and attack tolerance of complex networks. *Nature*, 406:378–382, 2000.

J. Allen. The Architecture of the Common Indexing Protocol (CIP). Internet Draft draft-ietf-find-cip-arch-01.txt, Internet Engineering Task Force, Nov. 1997.

J. Allen and M. Mealling. MIME object definitions for the Common Indexing Protocol (CIP). Internet Draft draft-ietf-find-cip-mime-02.txt, Internet Engineering Task Force, Jan. 1998.

K. M. Anderson, R. N. Taylor, and E. J. Whitehead, Jr. A critique of the open hypermedia protocol. *Journal of Digital Information*, 1(2), Jan. 1998.

B. Awerbuch. Optimal distributed algorithms for minimum weight spanning tree, counting, leader election, and related problems. In *Proceedings of the Nineteenth Annual ACM Conference on Theory of Computing*, pages 230–240, 1987.

P. G. Baker, A. Brass, S. Bechhofer, C. Goble, N. Paton, and M. Quinna. Transparent access to multiple biological information sources, an overview. Technical report, Department of Computer Science, University of Manchester, 1995.

A.-L. Barabási and R. Albert. Emergence of scaling in random networks. *Science*, 286(509), 1999.

A.-L. Barabási, R. Albert, and H. Jeong. Scale-free characteristics of random networks: The topology of the world wide web. *Physica A*, 281:69–77, 2000.

P. Barker. Use of the OSI directory for accessing bibliographic information. *Program: Electronic Libraries and Information Systems*, 26(2):345–359, 1992.

M. Barthélémy and L. A. N. Amaral. Small-world networks: Evidence for a crossover picture. *Physics Review Letters*, 82:3180–3183, 1999.

R. Bayardo, W. Bohrer, R. Brice, A. Cichocki, G. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. InfoSleuth: Agent-based semantic integration of information in open and dynamic environments. Technical Report MCC-INSL-088-96, Microelectronics and Computer Technology Corporation, Oct. 1996.

D. Beckett. IAFA Templates in use as Internet Metadata. In *Proceedings of the Fourth International World Wide Web Conference.* World Wide Web Consortium, Dec. 1995.

F. Bellifemine, A. Poggi, and G. Rimassa. Jade: A FIPA compliant agent framework. In *PAAM99 - The Fourth International Conference and Exhibition on the Practical Application of Intelligent Agents and Multi-Agents*, pages 97–108, 1999.

T. Berners-Lee, R. Cailliau, A. Luotonen, H. F. Nielsen, and A. Secret. The World Wide Web. *Communications of the Association for Computing Machinery*, 37 (8):76–82, Aug. 1994a.

T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol - HTTP/1.0. Request For Comments 1945, Internet Engineering Task Force, May 1996.

T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, May 2001.

T. Berners-Lee, L. Masinter, and M. McCahill. Uniform Resource Locators. Request For Comments 1738, Internet Engineering Task Force, Dec. 1994b.

D. Bertsekas and R. Gallaher. *Data Networks.* Prentice-Hall, 1987.

W. P. Birmingham. An agent-based architecture for digital libraries. *D-Lib Magazine*, July 1995.

D. Blacka, M. Kosters, M. Lu, L. Meador, M. Mealling, G. Pierce, A. Rao, J. Singh, S. Williamson, and K. Zeilstra. Referral Whois Protocol RWhois 2.0. Internet Draft draft-ietf-asid-rwhois-00.txt, Internet Engineering Task Force, Jan. 1998.

B. Bollobás. *Random Graphs.* Academic Press, 1985.

C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, and M. F. Schwartz. The Harvest Information Discovery and Access System. *Computer Networks and ISDN Systems*, 28(1-2):119–125, Dec. 1995a.

C. M. Bowman, P. B. Danzig, D. R. Hardy, U. Manber, M. F. Schwartz, and D. P. Wessels. Harvest: A scalable, customizable discovery and access system. Technical Report CU-CS-732-94, Department of Computer Science, University of Colorado, Mar. 1995b.

T. Bray, J. Paoli, and C. M. Sperberg-McQueen. Extensible Markup Language (XML) 1.0. W3C Recommendation REC-xml-19980210, World Wide Web Consortium, Feb. 1998.

F. Brazier, M. van Steen, and N. Wijngaards. On MAS scalability. In *Proceedings of the Second Workshop on Infrastructure for Agents, MAS and Scalable MAS at the Fifth International Conference on Autonomous Agents (ICMAS2001)*, pages 121–127, May 2001.

D. Brickley and R. Guha. Resource Description Framework (RDF) Schema Specification 1.0. Technical Report CR-rdf-schema-20000327, World Wide Web Consortium, Mar. 2000.

A. Broder, R. Kumar, F. Maghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web. In *Proceedings of the Ninth International World Wide Web Conference*. Elsevier Science, May 2000.

V. Bush. As we may think. *The Atlantic Monthly*, 176(1):101–108, 1945.

L. Carr, D. DeRoure, W. Hall, and G. Hill. The distributed link service: A tool for publishers, authors and readers. In *Proceedings of the Fourth International World Wide Web Conference*, pages 647–656, Dec. 1995.

I. Clarke. A distributed decentralised information storage and retrieval system. Ba dissertation, Division of Informatics, University of Edinburgh, 1999.

P. R. Cohen and H. J. Levesque. Intention is choice with commitment. *Artificial Intelligence*, 42:213–261, 1990a.

P. R. Cohen and H. J. Levesque. Performatives in a rationally based speech act theory. In *Proceedings of the 28th Annual Meeting of the Association for Computational Linguistics*, pages 79–88, 1990b.

CORBA. *CORBAservices: Common Object Services Specification.* Object Management Group, Mar. 1995.

T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* MIT Press, 1990.

Cyveillance. Internet exceeds 2 billion pages, 10 July 2000. URL `http://www.cyveillance.com/newsroom/pressr/000710.asp`. Press release.

J. Dale. *A Mobile Agent Architecture for Distributed Information Management.* PhD thesis, University of Southampton, Sept. 1997.

J. Dale and D. DeRoure. Towards a framework for developing mobile agents for managing distributed information resources. Technical report, Multimedia Research Group, Department of Electronics and Computer Science, University of Southampton, 1996.

R. Daniel. A trivial convention for using HTTP in URN resolution. Request For Comments 2169, Internet Engineering Task Force, June 1997.

R. Daniel and M. Mealling. Resolution of Uniform Resource Identfiers using the Domain Name System. Request For Comments 2168, Internet Engineering Task Force, June 1997.

W. H. Davies and P. Edwards. Agent-K: an integration of AOP and KQML. Technical Report AUCS/TR9406, Department of Computing Science, University of Aberdeen, 1994.

H. C. Davis, D. E. Millard, and S. Reich. Ohp - communicating between hypermedia aware applications. In J. Whitehead, editor, *Proceedings of the Workshop "Towards a New Generation of HTTP", A workshop on global hypermedia infrastructure, held in conjunction with the 7th International World Wide Web Conference*, Irvine, CA 92697-3425, Apr. 1998. University of California, Irvine Department of Information and Computer Science.

F. Dawson and T. Howes. vCard MIME directory profile. Request For Comments 2426, Internet Engineering Task Force, Sept. 1998.

DCMI. Dublin Core metadata element set, version 1.1: Reference description. Recommendation, Dublin Core Metadata Initiative, July 1999.

DCMI. Dublin core qualifiers. Recommendation, Dublin Core Metadata Initiative, 11 July 2000.

K. Decker, K. Sycara, and M. Williamson. Middle-agents for the internet. In *Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence (IJCAI-97)*, 1997.

K. Decker, M. Williamson, and K. Sycara. Matchmaking and brokering. In *Proceedings of the Second International Conference on Multi-Agent Systems (ICMAS-96)*, 1996.

D. C. Dennett. Intentional systems. *Journal of Philosophy*, 68(4), Feb. 1971.

D. DeRoure, S. El-Beltagy, N. Gibbins, L. Carr, and W. Hall. Integrating link resolution services using query routing. In *Fifth Workshop on Open Hypermedia Systems (OHS5)*, Darmstadt, Germany, Feb. 1999.

D. DeRoure, N. Walker, and L. Carr. Investigating link service infrastructures. In *Proceedings of the Eleventh International ACM Hypertext Conference*, 2000.

P. Deutsch, A. Emtage, M. Koster, and M. Stumpf. Publishing information on the Internet with anonymous FTP. Internet draft (expired), Internet Engineering Task Force, June 1995.

F. Donini, M. Lenzerini, D. Nardi, and A. Schaerf. Reasoning in description logics. In G. Brewka, editor, *Principles of Knowledge Representation and Reasoning*, Studies in Logic, Language and Information, pages 193–238. CSLI Publications, 1996.

N. Dushay, J. C. French, and C. Lagoze. Using query mediators for distributed searching in federated digital libraries. In *Proceedings of the Fourth ACM Conference on Digital Libraries (DL99)*, 1999. Submitted to ACM Digital Libraries 99.

ECS. *Handbook of the Department of Electronics and Computer Science.* Department of Electronics and Computer Science, University of Southampton, 1999.

J. Elliott and J. Ordille. Simple Nomenclator Query Protocol (SNQP). Request For Comments 2259, Internet Engineering Task Force, Jan. 1998.

P. Faltstrom, L. L. Daigle, and S. Newell. Architecture of the Whois++ service. Internet Draft draft-ietf-asid-whoispp-02.txt, Internet Engineering Task Force, Mar. 1998.

P. Faltstrom, R. Schoultz, and C. Weider. How to Interact with a Whois++ Mesh. Request For Comments 1914, Internet Engineering Task Force, Feb. 1996.

I. A. Ferguson and M. J. Wooldridge. Paying their way: Commercial digital libraries for the 21^{st} century. *D-Lib Magazine*, June 1997. Available online at `http://www.dlib.org/dlib/June97/zuno/06ferguson.html`.

R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol - HTTP/1.1. Request For Comments 2616, Internet Engineering Task Force, June 1999.

FIPA. FIPA 97 Specification part 1: Agent Management. Technical report, Foundation for Intelligent Physical Agents, 1997a.

FIPA. FIPA 97 Specification part 2: Agent Communication Language. Technical report, Foundation for Intelligent Physical Agents, 1997b.

A. M. Fountain, W. Hall, I. Heath, and H. C. Davis. Microcosm: An open model for hypermedia with dynamic linking. In *Proceedings of the European Conference on Hypertext.* Cambridge University Press, 1990.

J. French, A. Powell, and W. Creighton, III. Efficient searching in distributed digital libraries. In *Proceedings of the Third ACM Conference on Digital Libraries (DL98)*, pages 283–284, 1998.

J. C. French and C. L. Viles. Ensuring retrieval effectiveness in distributed digital libraries. *Journal of Visual Communication and Image Representation*, 7(1):61–73, Mar. 1996.

O. Frieder, D. A. Grossman, A. Chowdhury, and G. Frieder. Efficiency considerations for scalable information retrieval servers. *Journal of Digital Information*, 1 (5), Dec. 1999.

N. Fuhr. Resource discovery in distributed digital libraries. In *Digital Libraries '99: Advanced Methods and Technologies, Digital Collections*, 1999.

B. Ganter and R. Wille. Applied lattice theory: Formal concept analysis. In G. A. Grätzer, editor, *General lattice theory*, pages 591–605. Birkhäuser Verlag, second edition, 1998.

L. Gasser. MAS infrastructure definitions, needs and prospects. In *Proceedings of the First Workshop on Infrastructure for Agents, MAS and Scalable MAS at the Fourth International Conference on Autonomous Agents (ICMAS2000)*, 2000.

M. R. Genesereth and R. E. Fikes. Knowledge Interchange Format Version 3.0 Reference Manual. Technical Report Logic-92-1, Computer Science Laboratory, Stanford University, June 1992.

M. R. Genesereth, A. M. Keller, and O. Duschka. Infomaster: An information integration system. In *Proceedings of the 1997 ACM SIGMOD International Conference of the Management of Data*, May 1997.

N. Gibbins. Agent-based Resource Discovery in Distributed Information Systems. Master's thesis, Department of Artificial Intelligence, University of Edinburgh, 1997.

N. Gibbins and W. Hall. Scalability issues for query routing service discovery. In *Proceedings of the Second Workshop on Infrastructure for Agents, MAS and Scalable MAS at the Fifth International Conference on Autonomous Agents (ICMAS2001)*, pages 209–217, May 2001.

C. Goble and L. Carr. COHSE: Informed WWW link navigation using ontologies. In *Proceedings of the IEEE Colloquium "Lost in the Web"*, Sept. 1999.

R. Godin, R. Missaoui, and H. Alaoui. Incremental concept formation algorithms based on Galois (concept) lattices. *Computational Intelligence*, 11(2):246–267, 1995.

C. F. Goldfarb. *The SGML Handbook*. Oxford University Press, 1990.

Google. Google launches world's largest search engine, 26 June 2000. URL `http://www.google.com/pressrel/pressrelease26.html`. Press release.

S. Goose, J. Dale, W. Hall, and D. DeRoure. Microcosm TNG: A distributed architecture to support reflexive hypermedia applications. In M. Bernstein, L. Carr, and K. Østerbye, editors, *Proceedings of the Eighth International ACM Hypertext Conference*, pages 226–227. ACM Press, 1997.

M. Gorman and P. W. Winkler, editors. *Anglo-American Cataloguing Rules.* Library Association Publishing, second edition, 1988.

L. Gravano, K. Chang, H. García-Molina, C. Lagoze, and A. Paepcke. STARTS: Stanford protocol proposal for Internet retrieval and search. Technical report, Digital Library Project, Stanford University, Jan. 1997.

G. Grimmett. *Percolation.* Springer–Verlag, second edition, 1999.

T. R. Gruber. Toward principles for the design of ontologies used for knowledge sharing. In N. Guarino and R. Poli, editors, *Formal Ontology in Conceptual Analysis and Knowledge Representation.* Kluwer Academic Publishers, 1994.

F. Halasz and M. Schwartz. The Dexter Hypertext Reference Model. In J. Moline, D. Benigni, and J. Baronas, editors, *Proceedings of the NIST Hypertext Standardisation Workshop*, pages 95–133. US Government Printing Office, 16-18 Jan. 1990.

M. Harchol-Balter, T. Leighton, and D. Lewin. Resource discovery in distributed networks. In *Proceedings of the Eighteenth Annual ACM Symposium on Principles of Distributed Computing*, pages 229–237, 1999.

D. R. Hardy, M. F. Schwartz, and D. Wessels. *Harvest User's Manual.* University of Colorado, Version 1.4 patchlevel 2 edition, Jan. 1996.

K. Harrenstien, M. Stahl, and E. Feinler. NICNAME/WHOIS. Request For Comments 954, Internet Engineering Task Force, Oct. 1985.

R. Hedburg and P. Pomes. The ccso nameserver (ph) architecture. Request For Comments 2378, Internet Engineering Task Force, Sept. 1998.

C. Hedrick. Routing information protocol. Request For Comments 1058, Internet Engineering Task Force, 1988.

M. N. Huhns and M. P. Singh. The agent test. *IEEE Internet Computing*, 12(5): 78–79, Sept.-Oct. 1997.

ISO. Information processing – text and office systems – Standard Generalized Markup Language (SGML). International Standard ISO 8879:1986, JTC1/SC18, International Standards Organisation, 1986.

ISO. Hytime: Hypermedia/time-based structuring language. International Standard ISO 10744, JTC1/SC18, International Standards Organisation, 1997.

ITU. Information Technology – Open Systems Interconnection – The directory: Overview of concepts, models and services. ITU Recommendation X.500, International Telecommunication Union, Nov. 1993a.

ITU. Information Technology – Open Systems Interconnection – The Directory: Selected attribute types. ITU Recommendation X.520, International Telecommunication Union, Nov. 1993b.

ITU. Information Technology – Open Systems Interconnection – The Directory: Selected object classes. ITU Recommendation X.521, International Telecommunication Union, Nov. 1993c.

A. Jain, M. Murty, and P. Flynn. Data clustering: a review. *ACM Computing Surveys*, 31(3):264–323, 1999.

N. Jennings. Agent-based software engineering. *Artificial Intelligence*, 117(2):277–296, 2000.

M. A. Jovanovic, F. S. Annexstein, and K. A. Berman. Scalability issues in large peer-to-peer networks - a case study of gnutella. Technical report, Department of Electrical and Computer Engineering and Computer Science, University of Cincinnati, 2001.

KAG. An overview of KQML: A knowledge query and manipulation language. Draft, KQML Advisory Group, mar 1992.

F. Kappe. *Aspects of a Modern Multi-Media Information System.* PhD thesis, Institute for Foundations of Information Processing and Computer Supported New Nedia (IICM), Graz University of Technology, 1991.

F. Kappe. A Scalable Architecture for Maintaining Referential Integrity in Distributed Information Systems. Technical report, IICM, Graz University of Technology, 1994.

G. J. Klir and B. Yuan. *Fuzzy Sets and Fuzzy Logic: Theory and Applications.* Prentice-Hall, 1995.

J. P. Knight and M. Hamilton. Overview of the ROADS Software. Technical Report LUT SC-TR 1010, Loughborough University of Technology, 1996.

J. P. Knight and M. Hamilton. Dublin Core qualifiers. Draft, Loughborough University of Technology, Feb. 1997.

M. Koster. Guidelines for robot writers, 1994. URL `http://info.webcrawler.com/mak/projects/robots/guidelines.html`.

T. Krauskopf, J. Miller, P. Resnick, and W. Treese. PICS label distribution label syntax and communication protocols. W3C Recommendation REC-PICS-labels, World Wide Web Consortium, Oct. 1996.

Y. Labrou and T. Finin. A proposal for a new KQML specification. Technical Report CS-97-03, University of Maryland Baltimore County, Feb. 1997.

O. Lassila and R. Swick. Resource Description Framework (RDF) model and syntax specification. Technical Report REC-rdf-syntax, World Wide Web Consortium, 22 Feb. 1999.

P. Lauder, R. Kummerfeld, and A. Fekete. Hierarchical network routing. In *Proceedings of TriComm'91, the IEEE Conference on Communications Software*, pages 105–114, 1991.

S. Lawrence and C. L. Giles. Searching the World Wide Web. *Science*, 280(5360): 98, 1998.

S. Lawrence and C. L. Giles. Accessibility of information on the web. *Nature*, 400: 107–109, July 1999.

M. Lejter and T. Dean. A framework for the development of multiagent architectures. *IEEE Expert*, 11(6):47–59, 1996.

D. B. Lenat. CYC: a large-scale investment in knowledge infrastructure. *Communications of the Association for Computing Machinery*, 38(11):33–38, 1995.

Y. Lin, J. Xu, E.-P. Lim, and W.-K. Ng. ZBroker: A query routing broker for Z39.50 databases. Available from `www.arXiv.org` as cs.DL/9902018, 1999.

L. Liu. Query routing in large-scale digital library systems. In *Proceedings of the 15th International Conference on Data Engineering (ICDE'99)*. IEEE Press, Mar. 1999.

C. Lunau, P. Miller, and W. E. Moen. The Bath Profile: An international Z39.50 specification for library applications and resource discovery. Technical report, The Bath Group, June 2000. Available online at `http://www.ukoln.ac.uk/interop-focus/bath/`.

N. A. Lynch. *Distributed Algorithms.* Morgan Kaufmann, 1996.

P. Maes. Agents that reduce work and information overload. *Communications of the Association for Computing Machinery*, 37(7):31–40, July 1994.

MARBI/ALA/LOC. USMARC Formats: Background and Principle. Technical report, ALCTS/LITA/RUSA Machine-Readable Bibliographic Information Committee (MARBI) and Network Development and MARC Standards Office, American Library Association and Library of Congress, Nov. 1996.

C. D. Martin and J. M. Reagle, Jr. An alternative to government regulation and censorship: Content advisory systems for the Internet. Technical report, Recreational Software Advisory Council, 1996.

D. E. Millard, L. Moreau, H. C. Davis, and S. Reich. FOHM: A fundamental open hypertext model for investigating interoperability between hypertext domains. In *Proceedings of the Eleventh International ACM Hypertext Conference*, 2000.

J. Miller, P. Resnick, and D. Singer. Rating services and rating systems (and their machine readable descriptions). W3C Recommendation REC-PICS-services, World Wide Web Consortium, Oct. 1996.

S. Mizzaro. How many relevances in information retrieval? *Interacting With Computers*, 10(3):305–322, 1998.

R. Moats. URN syntax. Request For Comments 2141, Internet Engineering Task Force, May 1997.

P. Mockapetris. Domain names - concepts and facilities. Request For Comments 1034, Internet Engineering Task Force, Nov. 1987a.

P. Mockapetris. Domain names - implementation and specification. Request For Comments 1035, Internet Engineering Task Force, Nov. 1987b.

R. C. Moore. A formal theory of knowledge and action. In J. F. Allen, J. Hendler, and A. Tate, editors, *Readings in Planning*, pages 480–519. Morgan Kaufmann Publishers, San Mateo, CA, 1990.

L. Moreau, N. Gibbins, D. DeRoure, S. El-Beltagy, W. Hall, G. Hughes, D. Joyce, S. Kim, D. Michaelides, D. Millard, S. Reich, R. Tansley, and M. Weal. SoFAR with DIM agents: An agent framework for distributed information management. In *PAAM2000 - The Fifth International Conference and Exhibition of the Pratical Applications of Intelligent Agents and Multi-Agents*, pages 369–388, 2000.

J. Moy. OSPF version 2. Request For Comments 1247, Internet Engineering Task Force, July 1991.

M. N. Murty and A. K. Jain. Knowledge-based clustering scheme for collection management and retrieval of library books. *Pattern Recognition*, 28(7):949–963, 1995.

T. H. Nelson. Literary machines. Published by the author, 1987.

T. H. Nelson. Personal interview, Aug. 2001.

G. Neufeld. Descriptive names in x.500. In *Proceedings of the 1989 ACM Symposium on Communications Architectures and Protocols*, pages 64–71, 1989.

P. J. Nürnberg and J. J. Leggett. A vision for open hypermedia systems. *Journal of Digital Information*, 1(2), Jan. 1998.

H. Nwana, D. Ndumu, L. Lee, and J. Collis. Zeus: A tool-kit for building distributed multi-agent systems. *Applied Artificial Intelligence Journal*, 13(1):129–186, 1999.

OMG. *The Common Object Requst Broker Architecture.* Object Management Group, 1996.

J. Ordille. Internet Nomenclator Project. Request For Comments 2258, Internet Engineering Task Force, Jan. 1998.

J. J. Ordille and B. P. Miller. Distributed active catalogs and meta-data caching in descriptive name services. In *Proceedings of the IEEE International Conference on Distributed Computing Systems*, pages 120–129, May 1993.

R. Patil, R. Fikes, P. Patel-Schneider, et al. DARPA Knowledge Sharing Effort Progress Report. In W. S. Charles Rich, Bernhard Nebel, editor, *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference.* Morgan Kaufmann, 1992.

G. S. Pedersen. A browser for bibliographic information retrieval based on an application of lattice theory. In *Proceedings of the Sixteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 270–279, 1993.

J. E. Pitkow and R. K. Jones. Supporting the web: A distributed hyperlink database system. In *Proceedings of the Fifth International World Wide Web Conference*, May 1996.

S. Poslad, P. Buckle, and R. Hadingham. The FIPA-OS agent platform: Open source for open standards. In *PAAM2000 - The Fifth International Conference and Exhibition of the Pratical Applications of Intelligent Agents and Multi-Agents*, pages 355–368, 2000.

D. Raggett. HTML3.2 reference specification. W3C Recommendation REC-html32, World Wide Web Consortium, Jan. 1997.

A. Rao and M. Georgeff. Modeling rational agents within a BDI architecture. In R. Fikes and E. Sandewall, editors, *Proceedings of Knowledge Representation and Reasoning (KR&R-91)*, pages 473–484, San Mateo, CA, 1991. Morgan Kaufmann Publishers.

E. S. Raymond. *The Cathedral and the Bazaar.* O'Reilly & Associates, Inc., 1999.

S. Reich, U. K. Wiil, P. J. Nürnberg, H. C. Davis, K. Grønbæk, K. M. Anderson, D. E. Millard, and J. M. Haake. Addressing interoperability in open hypermedia: The design of the Open Hypermedia Protocol. *The New Review of Hypermedia*, 5:207–248, Apr. 2000.

P. Resnick and J. Miller. PICS: Internet access controls without censorship. *Communications of the Association for Computing Machinery*, 39(10):87–93, 1996.

J. Ritter. Why Gnutella can't scale. No, really., 2001. URL `http://www.darkridge.com/~jpr5/doc/gnutella.html`.

M. Rose. Directory assistance service. Request For Comments 1202, Internet Engineering Task Force, Feb. 1991.

M. Roszkowski and C. Lukas. A distributed architecture for resource discovery using metadata. *D-Lib Magazine*, June 1998. ISSN 1082-9873.

M. Salampasis. *An Agent-Based Hypermedia Digital Library.* PhD thesis, University of Sunderland, 1998.

M. Salampasis, J. Tait, and C. Bloor. Co-operative information retrieval in digital libraries. Presented in the 18th annual colloquium of the BCS IR SG, Manchester, U.K, Mar. 1996.

G. Salton. Mathematics and information retrieval. *Journal of Documentation*, 35 (1):1–29, Mar. 1979.

P. Sargent. Back to school for a brand new ABC. *The Guardian*, 12 Mar.:28, 12 Mar. 1992.

J. Searle. *Speech Acts: An Essay in the Philosophy of Language.* Cambridge University Press, 1969.

O. Shehory. A scalable agent location mechanism. In *Intelligent Agents VI - Proceedings of the Sixth International Workshop on Agent Theories, Architectures and Languages (ATAL'99)*, volume 1757 of *Lecture Notes in Artificial Intelligence*, pages 162–172, 1999.

A. P. Sheth and J. A. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, 1990.

Y. Shoham. Agent-oriented Programming. *Artificial Intelligence*, 60:51–92, 1993.

G. Singh. Leader election in complete networks. *SIAM Journal on Computing*, 26 (3):772–785, 1997.

M. P. Singh. Agent communication languages: Rethinking the principles. *IEEE Computer*, pages 40–47, Dec. 1998.

D. Skuce and I. Monarch. Ontological issues in knowledge base design: some problems and suggestions. In *Fifth Knowledge Acquisition for Knowledge Based Systems Workshop*, Banff, 1990.

K. Sollins. Architectural principles of Uniform Name Resolution. Request For Comments 2276, Internet Engineering Task Force, Jan. 1998.

K. Sparck Jones and P. Willett, editors. *Readings in Information Retrieval.* Morgan Kaufmann, 1997.

U. Straccia. A fuzzy description logic. In *Proceedings of AAAI-98, Fifteenth National Conference on Artificial Intelligence*, pages 594–599, 1998.

Jini Technology Core Platform Specification. Sun Microsystems Inc, Oct. 2000.

NBS. *UKMARC Manual.* The British Library National Bibliographic Service, fourth edition, 1996.

University of Michigan. *The SLAPD and SLURPD Administrators Guide Release 3.3*, 1996. Available online at `http://www.umich.edu/~dirsvcs/ldap/doc/guides/slapd/`.

M. Uschold and M. King. Towards a methodology for building ontologies. In *Workshop on Basic Ontological Issues in Knowledge Sharing, held in conjuction with IJCAI-95*, July 1995. AIAI-TR-183.

R. M. van Eijk, F. S. de Boer, W. van der Hoek, and J.-J. C. Meyer. A modal logic for network topologies. In M. Ojeda-Aciego, M. de Guzman, G. Brewka, and L. Pereira, editors, *Proceedings of the Seventh European Workshop on Logics in Artificial Intelligence (JELIA2000)*, pages 269–283. Springer–Verlag, 2000.

C. van Rijsbergen. *Information Retrieval.* Butterworths, second edition, 1979.

W3C. XML Linking Language (XLink). W3C Working Draft WD-xlink-19990726, World Wide Web Consortium, 26 July 1999a.

W3C. XML Pointer Language (XPointer). W3C Working Draft WD-xptr-19990709, World Wide Web Consortium, 9 July 1999b.

M. Wahl, T. Howes, and S. Kille. Lightweight Directory Access Protocol (v3). Request For Comments 2251, Internet Engineering Task Force, Dec. 1997.

D. J. Watts. *Small Worlds: the dynamics of networks between order and randomness.* Princeton University Press, 1999.

D. J. Watts and S. H. Strogatz. Collective dynamics of 'small-world' networks. *Nature*, 393:440–442, 1998.

C. Weider, J. Fullton, and S. Spero. Architecture of the Whois++ Index Service. Request For Comments 1913, Internet Engineering Task Force, Feb. 1996.

C. Welty. The ontological nature of subject taxonomies. In N. Guarino, editor, *Formal Ontology in Information Systems: Proceedings of FOIS'98*, Frontiers in AI Applications Series. IOS Press, 6-8 June 1998.

C. A. Welty and J. Jenkins. Formal ontology for subject. *Journal on Data and Knowledge Engineering*, 31:155–181, 1999.

U. K. Wiil and J. J. Leggett. The HyperDisco approach to open hypermedia systems. In *Proceedings of the Seventh International ACM Hypertext Conference*, pages 140–148, Mar. 1996.

S. Williamson, M. Kosters, D. Blacka, J. Singh, and K. Zeilstra. Referral Whois (RWhois) Protocol v1.5. Request For Comments 2167, Internet Engineering Task Force, June 1997.

M. Wooldridge and N. R. Jennings. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10(2), June 1995.

W. Yeong, T. Howes, and S. Kille. X.500 Lightweight Directory Access Protocol. Request For Comments 1487, Internet Engineering Task Force, July 1993.

W. Yeong, T. Howes, and S. Kille. Lightweight Directory Access Protocol. Request For Comments 1777, Internet Engineering Task Force, Mar. 1995.

Z39.50 Maintenance Agency. Information Retrieval (Z39.50): Application Service Definition and Protocol Specification. Standard ANSI/NISO Z39.50-1995, ANSI/NISO, July 1995.