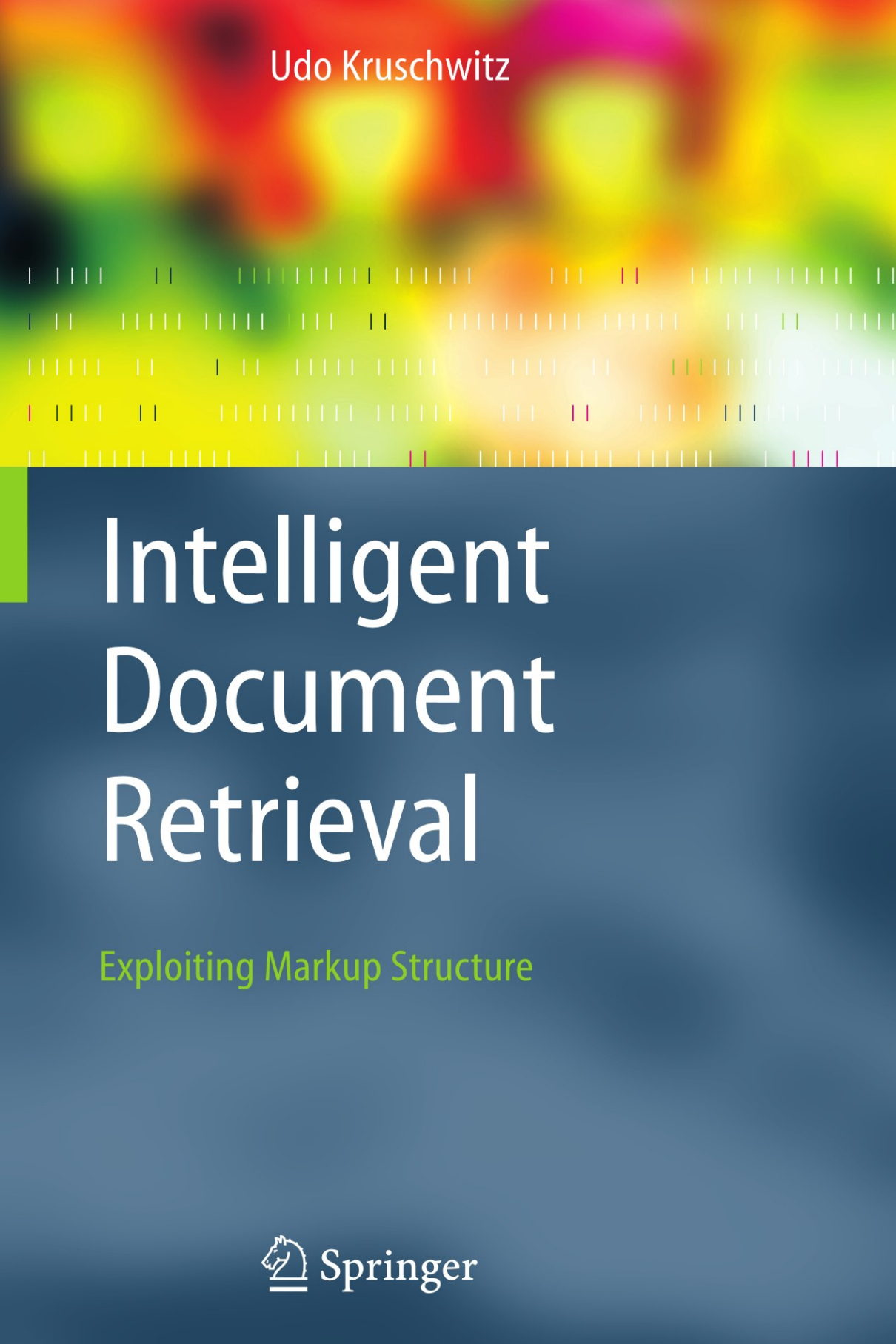
Udo Kruschwitz
Intelligent Document Retrieval
Exploiting Markup Structure
Springer

Intelligent Document Retrieval

THE SPRINGER INTERNATIONAL SERIES ON INFORMATION RETRIEVAL

Series Editor:
W. Bruce Croft
University of Massachusetts, Amherst

Also in the Series:

INFORMATION RETRIEVAL SYSTEMS: ***Theory and Implementation,*** *by Gerald Kowalski;* ISBN: 0-7923-9926-9

CROSS-LANGUAGE INFORMATION RETRIEVAL, *edited by Gregory Grefenstette;* ISBN: 0-7923-8122-X

TEXT RETRIEVAL AND FILTERING: ***Analytic Models of Performance***, *by Robert M. Losee;* ISBN: 0-7923-8177-7

INFORMATION RETRIEVAL: UNCERTAINTY AND LOGICS: ***Advanced Models for the Representation and Retrieval of Information*****,** *by Fabio Crestani, Mounia Lalmas, and Cornelis Joost van Rijsbergen;* ISBN: 0-7923-8302-8

DOCUMENT COMPUTING: ***Technologies for Managing Electronic Document Collections***, *by Ross Wilkinson, Timothy Arnold-Moore, Michael Fuller, Ron Sacks-Davis, James Thom, and Justin Zobel;* ISBN: 0-7923-8357-5

AUTOMATIC INDEXING AND ABSTRACTING OF DOCUMENT TEXTS, *by Marie-Francine Moens*; ISBN 0-7923-7793-1

ADVANCES IN INFORMATIONAL RETRIEVAL: ***Recent Research from the Center for Intelligent Information Retrieval***, *by W. Bruce Croft*; ISBN 0-7923-7812-1

INFORMATION RETRIEVAL SYSTEMS: ***Theory and Implementation,*** **Second Edition**, *by Gerald J. Kowalski and Mark T. Maybury;* ISBN: 0-7923-7924-1

PERSPECTIVES ON CONTENT-BASED MULTIMEDIA SYSTEMS, *by Jian Kang Wu; Mohan S. Kankanhalli;Joo-Hwee Lim;Dezhong Hong;* ISBN: 0-7923-7944-6

MINING THE WORLD WIDE WEB: ***An Information Search Approach***, *by George Chang, Marcus J. Healey, James A. M. McHugh, Jason T. L. Wang;* ISBN: 0-7923-7349-9

INTEGRATED REGION-BASED IMAGE RETRIEVAL, *by James Z. Wang;* ISBN: 0-7923-7350-2

TOPIC DETECTION AND TRACKING: Event-based Information Organization, *edited by James Allan;* ISBN: 0-7923-7664-1

LANGUAGE MODELING FOR INFORMATION RETRIEVAL, *edited by W. Bruce Croft, John Lafferty;* ISBN: 1-4020-12160-0

MACHINE LEARNING AND STATISTICAL MODELING APPROACHES TO IMAGE RETRIEVAL, *by Yixin Chen, Jia Li, James Z. Wang;* ISBN: 1-4020-8034-4

INFORMATION RETRIEVAL: ***Algorithms and Heuristics,*** *by David A. Grossman, Ophir Frieder;* ISBN: 1-4020-3004-5

INFORMATION RETRIEVAL: ***Algorithms and Heuristics,*** *by David A. Grossman, Ophir Frieder;* ISBN: 1-4020-3003-7

CHARTING A NEW COURSE: NATURAL LANGUAGE PROCESSING AND INFORMATION RETRIEVAL: ***Essays in Honour of Karen Sparck Jones,*** *edited by John I. Tait;* ISBN: 1-4020-3343-5

Intelligent Document Retrieval

Exploiting Markup Structure

by

Udo Kruschwitz
University of Essex,
Colchester, U.K.

Springer

A C.I.P. Catalogue record for this book is available from the Library of Congress.

ISBN-10 1-4020-3767-8 (HB)
ISBN-13 978-1-4020-3767-2 (HB)
ISBN-10 1-4020-3768-6 (e-book)
ISBN-13 978-1-4020-3768-9 (e-book)

Published by Springer,
P.O. Box 17, 3300 AA Dordrecht, The Netherlands.

www.springeronline.com

Printed on acid-free paper

Printed in the Netherlands.

Contents

Foreword

Udo Kruschwitz's book, based on his PhD thesis, argues that for Google-type web searches on limited domains, or on site-specific intranets, performance can be enhanced by making use of a domain model of the entities and relations characteristic of that site. He shows how to use document structure mark-up and lexical co-occurrences within and across documents to construct such domain models automatically. Users are then able to engage in a dialogue in which cues are provided, based on the domain model, to enable them to relax or to refine their search until they find what they are looking for.

The method has been implemented and embodied in two different practical applications, and these have been evaluated in user trials. These trials provide some evidence that the technique is effective in helping users.

The research carried out by Udo Kruschwitz and reported in this book is a model of how to combine computational linguistics and information retrieval techniques in a theoretically motivated - but practical - application, which has also been fielded and empirically tested. Anyone working in the fields of computational linguistics, information retrieval, document summarisation, web searching or question answering will find something of value in this book.

Oxford,
23^{rd} March 2005

Stephen Pulman
Professor of General Linguistics
Oxford University

Preface

Thanks to everyone who helped me with this book.

I wish to thank Sam Steel, Anne De Roeck, Massimo Poesio, Mounia Lalmas, Thomas Rölleke, Nick Webb, Paul Scott, Ray Turner, Maria Fasli, Stephen Pulman and Bill Black in particular as well as all the students and colleagues who volunteered to help with the evaluations. Some of the evaluation work described in here was joint work with Hala Al-Bakour supported by EPSRC grant GR/R92813/01, who I would like to thank as well. Thanks to Doug Arnold for suggesting a book title. Furthermore, I am very grateful to Robbert van Berckelaer at Springer for his assistance in preparing this book.

Finally, special thanks to my family over there in Germany, the German Society and the Horse & Groom.

Wivenhoe, *Udo Kruschwitz*
4^{th} April 2005

List of Figures

List of Tables

1

Introduction

> "The issue of extracting the structure of some text [...] is a challenging issue" [1].

We are witnessing a massive growth of electronic natural language resources. Most noticeable is the development of the *Web*, with online newspapers, product catalogues, data archives etc. Millions of users access the Web or other electronic document collections every day. In this book we look at a single aspect of this rather complex area: How can we help a user to navigate a document collection easily, and how can we assist a user who wants to search a collection for documents that satisfy some information need?

We will not look at general Web search, but instead we will concentrate on smaller collections such as Web sites or collections of classified advertisements. They represent much narrower domains unlike the broad coverage of the Web. One reason for considering this area a worthwhile research issue is the fact that searches in document collections often return either large numbers of matches or no suitable matches at all. We acknowledge that Web search algorithms have matured significantly over the past few years and that a search request submitted to *Google*[1] typically returns excellent matches for a user query. Nevertheless, this is not always the case if the collection is only a fraction the size of the Web and the documents cover a much smaller range of topics. Such collections are very common in institutions, universities or companies.

Some explicit knowledge about the domain, i.e. a *domain model*, could be useful to help the user find the right documents. A domain model could encode relations between words or phrases that have been extracted by analysing the document collection. The graph in Fig. 1.1 gives an idea of how such a model may be structured. We see a tree of related terms that can be used to either assist a user in the search process, perform automatic query refinements or allow the user to browse the collection. Note that the *types* of relation in the sample tree are not formally specified. This is significantly different from

[1] `http://www.google.com`

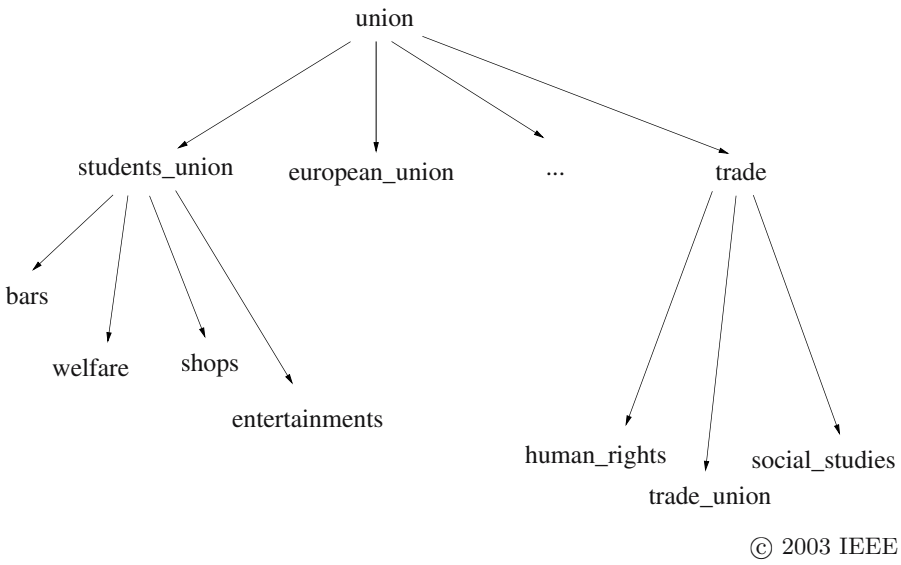

Fig. 1.1. Sample relations in a domain model

formal ontologies or lexical resources such as *WordNet* - a large dictionary-like resource originally developed for English that encodes linguistic relations between lexical items (e.g. synonymy, antonymy etc.) [50].

However, such a domain model is generally not available. Existing linguistic knowledge sources and ontologies tend to be either too generic (this is a problem with *WordNet*) or very knowledge-intensive and tailored specifically to some well-defined domains, e.g. [4, 60]. In other words, they do not reflect the particularities of a newly selected document collection and therefore are not the best candidates for assisting users in a search process. Furthermore, changes in the collection or an entirely new domain would require changes in the ontology, which can be a complex task [105].

What can be done about this? Ideally, we would like a domain model constructed on the fly in an automated fashion without assumptions about the documents' content, i.e. without having to *know* what the documents are about. We will discuss in detail how the documents' markup structure can be used to build such models.

Electronic documents typically have *some* internal structure. It could be HTML markup in which the documents are encoded; it could as well be some less obvious and more implicit structure. However, we are not interested in documents whose content has been fully semantically marked up. In other words, what we look at is the *automatic* processing of *partially structured* (or *semistructured*) data. The term *semistructured* data is not clearly defined in the literature. Semistructured data is sometimes even refered to as *unstruc-*

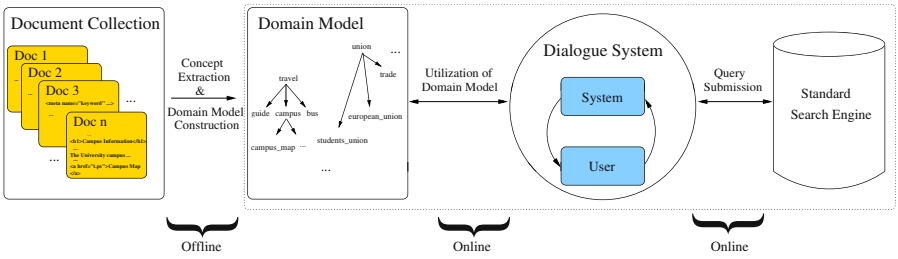

Fig. 1.2. Extraction and application of markup-based knowledge

tured data [24]. Soderland characterizes semistructured texts as texts that have "fairly stereotyped information but are not rigidly formatted" [143]. Henzinger et al. use the term *vaguely-structured data* to describe the data that is typically found on the Web [65]. We will adopt the flexible notion of data "that is neither raw data, nor very strictly typed as in conventional database systems" [1]. We will consider *vaguely-structured*, *partially structured* and *semistructured* data as synonymous terms in this context.

We will demonstrate how we can turn such document collections into usable domain models. The crucial idea is that a tremendous amount of implicit knowledge is stored in the markup of documents. But not much has been done to use this particular knowledge. We will exploit this knowledge to build domain models in order to assist a user in a search process. Pure markup can be used to build a domain model, but that is not enough when searching a document collection. It will be most useful in combination with a dialogue strategy. A simple dialogue system that has access to the automatically extracted domain knowledge will be a sensible tool to assist users in searching the document collection. Figure 1.2 is a simplified overview of the processes involved and discussed in this book.

To summarize, we can distinguish three major areas that this book will cover:

- We will show how markup structure can be used to build domain models rapidly and fully automatically.
- We will apply the knowledge that has been extracted in the first step. We will demonstrate how such domain models are able to assist a user by refining the choices as he or she searches the document collection. This will appear to the user as a specialized dialogue with the system.
- Finally, we will present implemented systems which can easily be adapted to new document collections.

1.1 Introductory Examples

This section will illustrate the research problem addressed by this book with the help of two example collections. A more formal account will be given later on.

The first document collection is the *University of Essex* Web site. One of our running examples in this sample collection will be the user query *"union"*, which is a frequent query according to the log files of the local search engine. The user might not be aware of the fact that the query is highly ambiguous. In fact, there are Web pages in that domain presenting information about the *trade union*, *students union*, the *European union* and a *Christian union*. Not just that, but there are a number of pages devoted to *discriminated unions*. That could well be the pages the user expects as an answer if that user is a student currently writing an assignment in the *Distributed Computing* module. A domain model that reflects these possible interpretations could be a useful aid to inform the user of what types of documents there are (or are not) in the space of possible answers. At the same time it would guide the user in the process of narrowing down the search by explicitly offering sensible choices. Obviously, such knowledge is typically not available as an off-the-shelf product because it will vary dramatically between different document collections. The graph in Fig. 1.1 could be part of a useful domain model. Applied to the *"union"* query it may trigger a system response such as displayed in Fig. 1.3.

The actual implementations that we will discuss later on vary from this system response in that they combine the information provided by the domain model with results of a standard search engine as can be seen in Fig. 1.4. For simplicity we will sometimes use the type of response in Fig. 1.3 elsewhere in the book.

Another query in this domain could be *"lecturers in AI"* for which the existing university search engine cannot find any result even though the information is there, but not explicitly on a single page. Again, we would like to see some suggestions based on the domain model that would allow us to locate documents we are interested in. In this particular case the system could propose query relaxations (e.g. suggest partial matches or some more general term for which matches can be found).

To cope with both types of queries we process the document collection by exploiting the HTML tags used to mark up the documents. As we will see later, we can utilize the different types of HTML markup contexts without trying to *understand* the documents. The result of this processing step is a simple domain model that can either be used to browse the collection or to assist a user searching for documents. The domain model is a set of term hierarchies which are compared against the user query, allowing the search system to offer suitable choices to constrain or relax the original query.

We will discuss both the construction of the domain model and the dialogue framework that applies such a model. One motivation for the use of a dialogue manager is that an automatically constructed domain model will

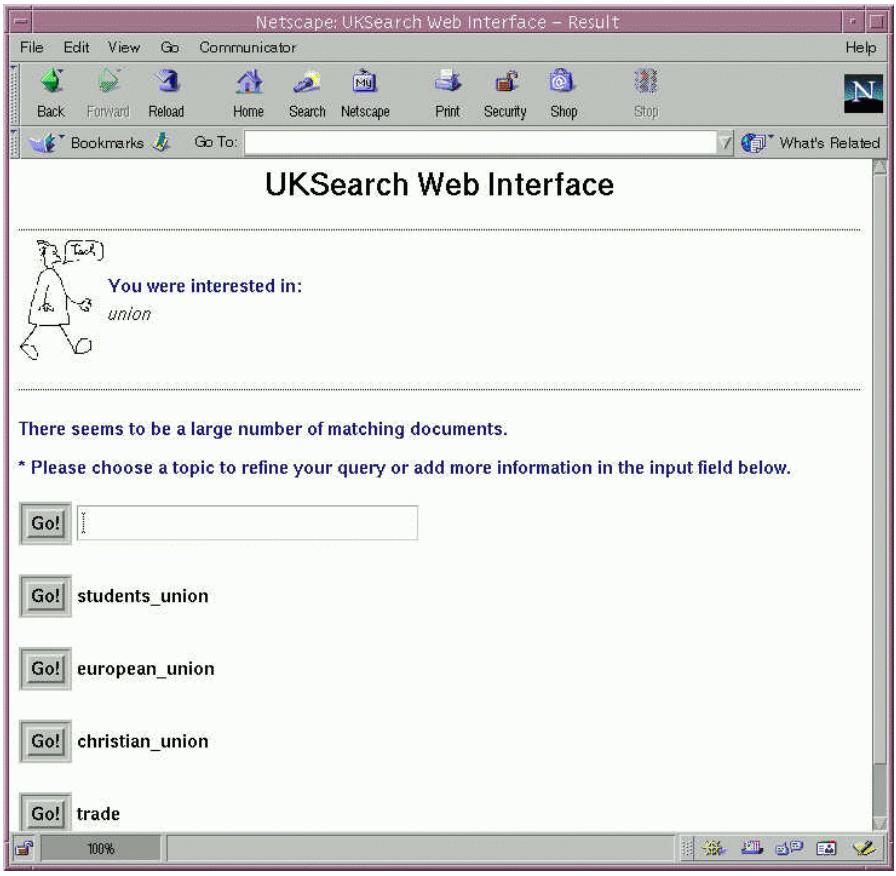

Fig. 1.3. Applying a domain model in ad hoc search

necessarily be incomplete. To use the last example: if a link between *lecturers* and *AI* cannot be uncovered, then a dialogue system may still be able to give the user some assistance as displayed in Fig. 1.5.

Let us briefly look at a very different document collection: a classified directory. Although this type of data sources will not be our main focus, it is a good example to demonstrate the flexibility of the methods introduced in this book.

An actual example collection are the local *Yellow Pages*[2]. Note that although we talk about a specific collection in this context, the issues discussed here are more general and can also be observed in other types of classified directories. The electronic version of the *Yellow Pages* we had access to is encoded as the so called *Yellow Pages data file* or *production tape* in electronic

[2] *Yellow Pages® and Talking Pages® are registered trade marks of Yell Limited in the United Kingdom.*

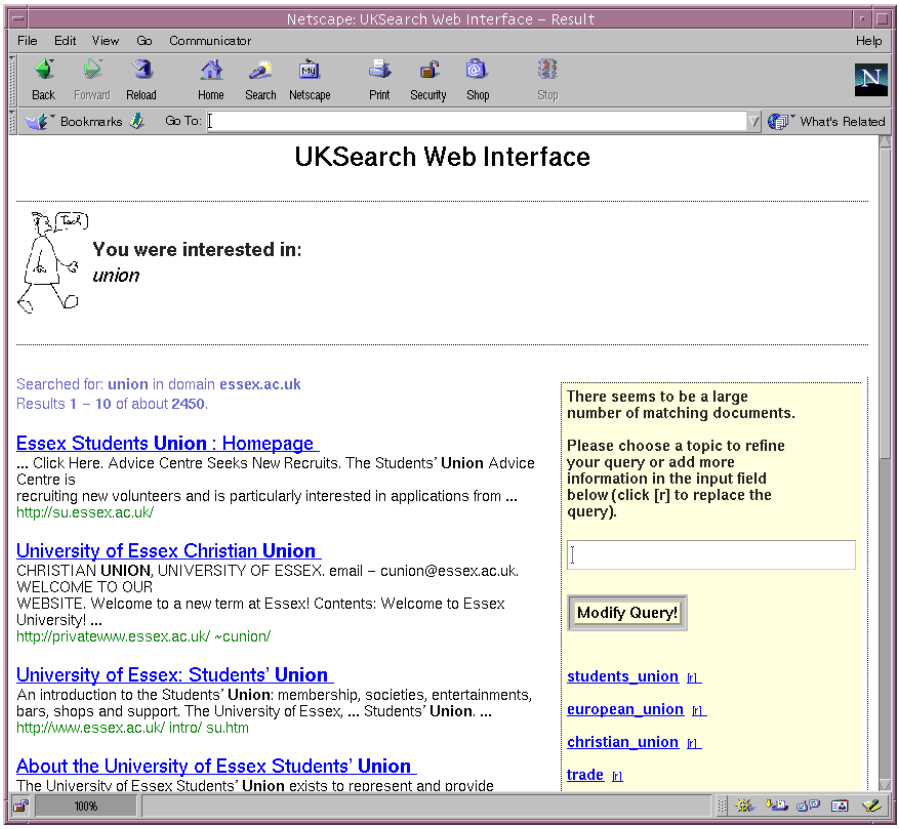

Fig. 1.4. Combining search engine results with the domain model

form for each area. It is an interesting data source that bears some resemblance with the documents of a Web site as seen in the first example in the sense that there is some structure, but much of it implicit rather than explicit. We find different types of advertisements that consist of business names, address information etc. However, in the actual *Yellow Pages data file* the markup tends to be fairly coarse-grained. The structure allows us to identify an individual advertisement and the name of the business, but there is no explicit markup that would tell us what part of this entry represents a telephone number, the address etc. Moreover, we typically find some *free text* which is an optional natural language portion to be printed in the advertisement along with the address as in the following example:

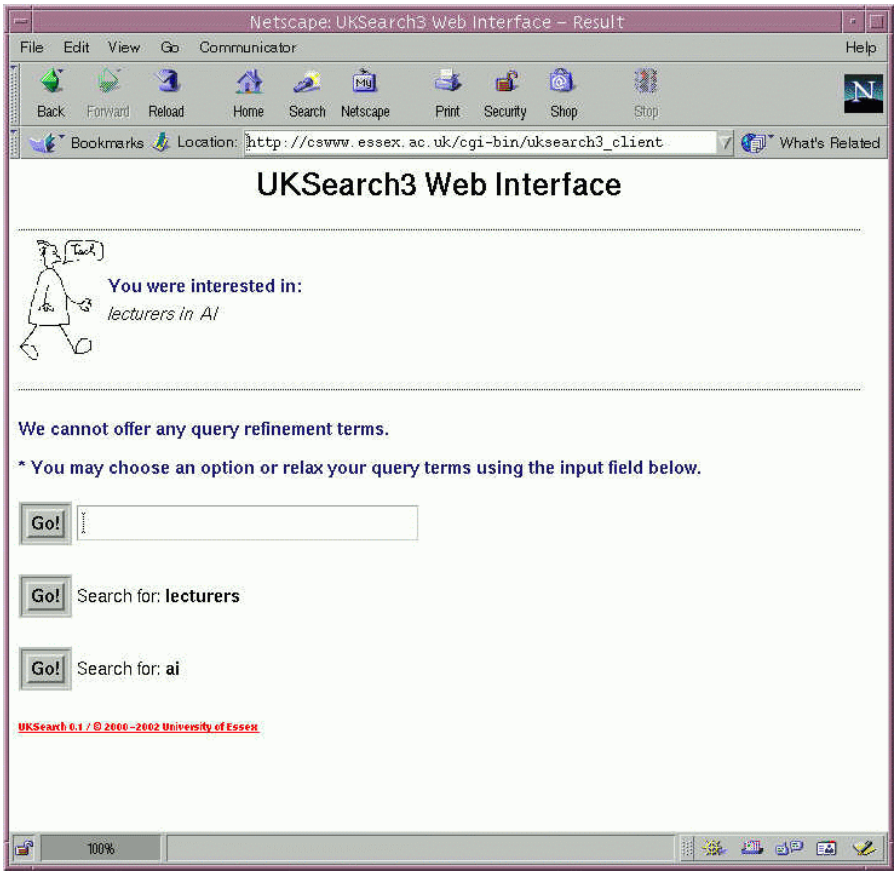

Fig. 1.5. Simple relaxation options proposed by the search system

Kruschwitz Golf & Leisure Wear

Suppliers Of All Top Brand Golf Equipment

100 High Street Colchester 822990

The point is that the markup structure is not rich enough to semantically encode the documents (i.e. advertisements in this context), but it makes it possible to distinguish different parts of a document (which could then be analyzed further). Although the advertisements are not encoded in some standard markup language, the document source can be abstracted in much the same way like Web documents, only that there are other types of markup and they are realized differently. Searching this data collection should be almost identical to searching the *University of Essex* Web site. In both cases we can have a very simple dialogue system as a mediator between user and database that has

access to the automatically processed data collection. However, when we look at collections of Web documents we will discuss in detail both the automatic construction of a domain model and its utilization in the search process. In the classified directory example the main focus of the application will be on the actual dialogue manager rather than the domain model construction. A reason for that is that - in contrast to Web documents - additional explicit knowledge sources (such as classification structures) are typically available in such collections which can be utilized in the model construction and in the search process.

To summarize, we are interested in processing and searching partially structured collections of natural language documents that can be characterized as being:

- more structured than free text
- less structured than traditional (relational) *Database Management Systems (DBMS)* or semantically encoded data sources such as XML documents
- typically not excessively large, possibly even rather small to apply standard *Information Retrieval (IR)* methods effectively.

We shall now look at the methodology we will apply in more detail. Later we will describe the extraction and application techniques more formally.

1.2 Using Markup to Extract Knowledge

> "The Semantic Web will bring structure to the meaningful content of Web pages, creating an environment where software agents roaming from page to page can readily carry out sophisticated tasks for users." [16]

Finding information in a large pool of documents is a task that researchers in the *Information Retrieval* community have been working on for half a century. For a long time the main paradigm has been to treat documents as a bag of words. The frequency of occurrences and the distribution of words within the document collection can then be used to approximate the documents most closely related to a user query.

However, the emergence of the Web has had a huge impact on this research for a number of reasons, including

- the fact that everybody can be an author and publish their work online
- the resulting massive growth of online data
- the use of markup languages that provide online documents with additional structure, and
- the growing interest in new search methods arising from this situation.

This book will not discuss how these observations can be used to improve search applications *in general* by indexing the documents in a more sophisticated way, nor will it discuss whether or not such a strategy is feasible or even useful for large scale search engines (the interested reader will find such issues discussed in much detail in [28]). We will instead focus on "narrower" domains, such as Web sites or other document collections. The reason is that there are many thousands such collections which are all different and we want to build domain models which reflect knowledge that is specific for the collection rather than general purpose knowledge. In addition, we want to be able to build such models rapidly.

We are still far away from Berners-Lee's vision of a *Semantic Web* [16]. Despite the fact that there is a constantly growing quantity of data sources containing explicitly encoded semantics, most data still comes with much less structure, usually encoding presentation details rather than content. However, we are witnessing a rapid growth in the number of specialized search engines alongside numerous systems restricted to a particular domain (e.g. company Web sites or intranets). The indexing processes described in this section (and in more detail later on) will concentrate on these smaller domains. In particular, it will show how the construction of a domain model on the fly can lead to a search engine tailored to a specific domain.

The indexing process which includes the automatic construction of a *domain model* is the fundamental step towards an "intelligent" search system. A good domain model could drive a user-friendly interaction which goes beyond simple keyword matching. Instead, by interacting with the user the system helps the user navigate through the possible matches until the appropriate documents are found. Alternatively, such a model can be used to browse the document collection.

Why can we not adopt existing structured knowledge sources and incorporate them in the search system? Entirely relying on domain-independent sources like *WordNet* has been shown to not be very suitable unless these sources are customized in an expensive manual process. We will discuss this issue later.

The idea of automatically acquiring domain models from documents is not new. Different approaches exist to preprocess the document collection to construct a model automatically. However, this research has so far focussed mainly on word co-occurrences or linguistic information, e.g. [134, 9, 99].

We will follow an alternative approach, which is to capture the semantic content of a document collection by exploiting the markup of the documents. Basically, the number of index terms is reduced dramatically by *selecting only words or phrases that are found in at least two different markup contexts in the same document*. We will call such terms *concepts* (or *conceptual terms*). This process can be performed automatically with no expert knowledge. It is largely independent of the actual language used in the document and more importantly it uncovers the structure of a collection of documents while itself being domain-independent. It should not be seen as an alternative to standard

search technologies such as those applied in Web search engines. Instead, there is great potential in the combination of state-of-the-art search mechanisms and the knowledge that we extract from the documents. When we introduce the implemented prototypes later on we will show how we merge the results that come back from a standard search engine with the knowledge uncovered in the domain model construction process.

Obviously, the term *concept* is heavily overloaded in the scientific world and here we introduce yet another interpretation. Nevertheless, it seems to be the most appropriate terminology in this context and we will use it consistently throughout the book.

To use a concrete example of how concepts are extracted we look at Web documents. For this type of documents we may want to define markup contexts based on frequently used HTML tags, such as the tags that identify the document title (i.e. `<title>`), anchor text (`<a>`), headings in the document (`<h1>`, `<h2>`, ...), text enclosed in meta tags (i.e. tags that the author of the document provides to describe the content of the Web page) etc. Any term that can be found in more than one of these contexts in a document will then be considered a concept. We may identify a handful of such terms for a particular document; for other documents we may not find any at all. Note that this concept identification process avoids the interpretation of *individual* markup contexts which makes the method much more generic than techniques for knowledge acquisition that rely on specific assumptions about the use of certain tags, commonly used patterns etc. as for example suggested in [103].

However, this is just one example, and we are by no means restricted to HTML documents. If we want to use the same methods to identify concepts in the different sections of this book, we may define markup contexts based on the font types used in the document. This would give us a number of different markup contexts (e.g. text in the default font, italicized font, bold font etc.) We would then treat the terms "concepts", "domain" and the compound "domain model" in this section of the book as concepts (they all appear in two markup contexts: in the default font and italicized font).

Apart from identifying concepts we introduce *related concepts*. Two concepts are related if they were both found to be concepts in the same document. In the above example we identified the terms "domain model" and "concepts" to be conceptual terms, but furthermore they represent a pair of related concepts.

The domain model (we will also use the term world model) we construct exploits the relationship between concepts. The result of the model construction process is a set of concept hierarchies. The hierarchies (part of one is displayed in Fig. 1.1) are applied in a dialogue system which will be discussed in more detail in the next section.

We should stress at this point that we adopted the term *hierarchy* and will use it throughout the book although one could argue that the structures we call hierarchies are in fact simple directed graphs. However, from a pragmatic point of view we interpret nodes in the graph as possible user queries (to be

discussed shortly), and the further away we move from the root node the more *specific* the queries get. Hence, we decided to call these graphs hierarchies.

As indicated, one of the main motivations for automatically acquiring a domain model in the first place is the inadequacy of a domain-independent knowledge source like *WordNet* to search Web sites or other document collections or specific domains of documents in the widest sense. This is true for at least the following reasons.

First of all, a number of relations that may be inherent in the documents of the domain will not be present in the knowledge source, i.e. relations that reflect *world knowledge* rather than linguistic information (cf. the *union* example in Sect. 1.1).

On the other hand, a large number of links in a domain-independent model will be entirely irrelevant to a specific domain. To stick to the *union* example, *WordNet* senses two (*the United States during the Civil War*), three (*the act of pairing a male and female for reproductive purposes*) and five (*state of being husband and wife*) are not relevant to the *University of Essex* sample domain at all, nor are some of the other senses. However, there is no way to decide a priori for a given document collection which senses are (or are not) relevant.

In addition to these problems, the domain may change over time which makes it desirable to construct a model on demand exploiting the actual data.

Furthermore, a domain model that is based on linguistic concepts will not be best suited for tasks like interactive search. A simple user study that investigated the usefulness of automatically created concept hierarchies in an *interactive query expansion* task found that far more than half of the expansion terms selected by users were terms conceptually related to the initial query [74]. For example, *tooth* and *dentist* are considered conceptually related but this type of relation is not defined in *WordNet*. Synonyms (i.e. knowledge that can be found in *WordNet*) were chosen much less frequently.

Despite all these concerns we have to accept that the *structure* of a knowledge source like *WordNet* looks very promising for the type of application we have in mind. The reasons include the clear and simple organization, its applicability to natural language engineering systems, and the advantage that the model does not have to be rebuilt every time the document collection needs to be re-indexed: The model is independent of the actual documents.

To make the last point clearer, compare this with a very different approach in which the domain knowledge is closely linked to the documents and not separated. The MIT *START* system[3] links knowledge to the actual documents [78, 79]. *START* is a question answering system that indexes a document collection by annotating documents and storing those annotations in a knowledge base. The online search system tries to match the user query against the annotations. These annotations essentially describe the questions that some part of a document is able to answer.

[3] `http://www.ai.mit.edu/projects/infolab/`

The interpretation of the concept hierarchies in our domain model is different from a number of other models. In *WordNet* for example the links between two sets of terms stand for clearly defined semantic relations (e.g. *hypernymy*, *antonymy* etc.). Noun compounds can also be classified according to the type of relation between compounds [127]. Furthermore, thesauri have been built for specialist domains to represent hypernym-hyponym relationships between terms, e.g. [3] (but the methods rely on specific linguistic cues). Other term hierarchies are constructed in a way that general terms are placed at the top levels, more specific terms are found further down [134].

A different approach is to use an existing subject taxonomy and enrich it by placing potential query terms into the appropriate categories as proposed in [34]. However, this approach relies on "constructing the hierarchy based mainly on the analysis of human observations"; in other words human effort is required to construct a taxonomy in the first place.

Our aim is *not* to capture the actual *semantic* relations that exist between concepts but the fact *that* there is some relation, one that can be used to guide a user in the search process. We assume that the concepts that we can extract in an indexing process simply based on markup structure are likely to be among those terms that users submit as real queries when they search the document collection. In fact, the log files of the queries submitted to the search engine in our *University of Essex* sample domain prove that this is a sensible assumption (to be discussed in more detail in Sect. 6.3).

We construct the domain model automatically in an offline process. The process is (as indicated earlier) based on our idea of *related concepts*. We shall briefly outline the model construction process here. Later on we will look at it in more detail.

Each hierarchy in the model consists of nodes that a concept represents (see Fig. 1.6). Exploiting the fact that concepts identified in the indexing process are likely to turn up as real user queries, the model-construction process is a sequence of user request simulations that does not use live user queries but rather the identified concepts. Each concept is a potential query for which a hierarchy is built. We start with a single concept as a possible user query (for example *language_linguistics* in Fig. 1.6). Using the relations detected in the source data, we can then explore all possible ways of constraining this query by adding a single concept to the query in a query modification step. The interesting terms that could be added in such a step are all concepts *related* to the original query term (concept). A new node is automatically created if there are any documents in the collection matching the refined query.

The model construction process is an iterative process that can be applied to the new queries until eventually we end up in leaf nodes - that is, nodes that typically represent very specific queries for which only a small set of documents can be found. In each of those iterative steps, we would expand the current query by a single concept related to *all* query terms collected so far. Roughly speaking, for each concept identified in the document collection we construct a term hierarchy as follows:

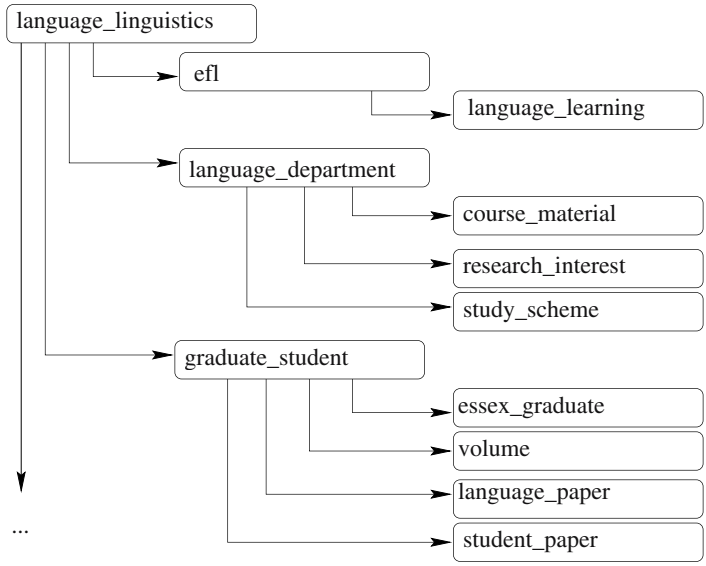

Fig. 1.6. Concept tree for the compound *language_linguistics*

1. Create a root node that contains the concept.
2. Pick a node, collect all concepts from root node down to the selected node. For each concept that is related to *all* these concepts, create a daughter node.
3. Perform step 2 until there are no unexplored nodes left.

Hence, once we have constructed a tree for a particular concept, then a path from the root node to any other node in this tree can be seen as adding new terms to a query. In other words, any node represents a cluster of documents in the collection: all those documents that match each of the concepts associated with the nodes passed by traversing the graph from the root to the current node.

As a result of this *offline* model construction process we get exactly the sort of structures that we are interested in: a set of trees with terms representing the nodes. These structures are not calculated at run time. Furthermore, in a relatively static domain we will not need to build a new domain model every time the collection is re-indexed.

Let us go back to the example. The compound *language_linguistics* is a concept that has been identified in the indexing process. Figure 1.6 displays part of the tree created with *language_linguistics* as the root node. Not all concepts related to *language_linguistics* are displayed, only the three most relevant ones. The most relevant related concept is *efl* which stands for "English as a Foreign Language". There is only one more concept related to both of the terms *language_linguistics* and *efl*, which is *language_learning*. This is the

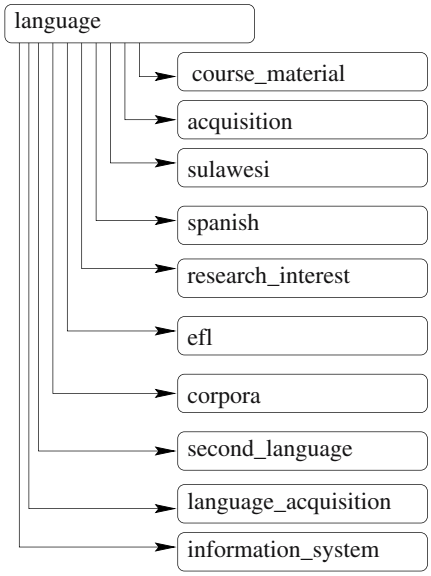

Fig. 1.7. Concept tree for the term *language*

end of the path. The construction process of the world model determined that there was no related concept that could be added to the three others which would describe a non-empty cluster of documents.

How does this relate to real user queries? Well, *"language and linguistics"* is a query frequently submitted to the search engine of the sample domain. This means the domain model can actually assist a user who submits the query.

Figure 1.7 depicts the top ten related concepts for *language* (*"languages"* is a query slightly less frequently submitted to the search engine than *"language and linguistics"*). Unlike in the first example we only display the first level (below the root node) of the tree. Naturally, the tree is much bigger than the one in the earlier example. As can be seen the related concepts are not necessarily related to linguistics. The third most important concept related to *language* is *sulawesi*. It turns out that this term does not refer to an Indonesian island, but to some framework in the context of wearable devices that deals with natural language understanding. Despite the fact that an obviously existing relation between the two terms *language* and *sulawesi* was discovered in the automatic knowledge extraction process, it is not immediately clear whether this is in fact useful in this context. This can only be established by observing real users' behaviour, to be discussed later in the book.

Before we outline how the extracted knowledge can actually be utilized we want to point out that what we said about our definition of the term *concept* (being heavily overloaded) is equally true for terms like *domain model*, *related*

concepts etc. The interpretation of each of these terms varies from one research community to another. We will formally define the terminology used in this book in Chaps. 3 and 5.

1.3 Applying the Extracted Knowledge

> "A tremendous amount of heterogenous information exists in electronic format (the most prominent example being the World Wide Web), but the potential of this large body of knowledge remains unrealized due to the lack of an effective information access method." [78].

We just discussed the issues involved in characterizing and processing the data sources, getting at the actual content of them and organizing the extracted knowledge in some simple models. This pre-processing step is essential for what is to follow, the construction of a simple specialized dialogue system that allows a user to search through the document collection. This dialogue system offers and makes choices about search through a set of documents based on the domain knowledge extracted automatically from the documents' markup structure.

Our approach inherits a number of ideas and motivations from elsewhere. For one, it has to be accepted that state-of-the-art search engines can handle a large percentage of user queries nicely. Therefore, we do not want to ignore established search technologies. Instead, we want to tackle the remaining percentage of queries that cannot be handled in the straightforward way as it is done by search engines.

A motivation for our approach of a simple dialogue system to search Web documents in particular is a user study concerning human computer interaction issues presented in [32]. The study comes to the conclusion that users prefer to see some categorized output rather than a simple list of ranked documents. It was shown to be easier for the users to get to the right documents quickly by using a hierarchical classification structure.

The type of model we construct can function as a directory-like structure similar to those found in classified directories like the *Yellow Pages* or manually maintained Web directories such as *Yahoo!*[4] or the *Open Directory*[5].

Knowledge sources created like this can then be utilised in applications like browsing, information retrieval, document clustering or classification. We concentrate on information retrieval, more specifically, ad hoc retrieval using a simple dialogue system. The domain model can be applied to guide a user through the term hierarchies as one would go through a directory structure: choosing some entry point and then navigate within this hierarchy in a *hierarchy-driven* fashion.

[4] `http://www.yahoo.com`

[5] `http://www.dmoz.org`

Earlier, when we discussed the domain model construction we actually explained how the *offline* construction process works. Concept terms are used to modify queries consisting of other concept terms and the results of that are encoded in the domain model. As long as the user submits queries that consist of terms identified as concepts in the indexing process, a simple lookup in the domain model will retrieve a number of query modifications without the need of any online processing. After all, the domain model is based around the idea of how to organize discriminating terms so that the domain model is a useful tool to be applied in the search task.

But when dealing with real user queries we will not be able to rely entirely on what is encoded in the domain model. We have no idea what queries are going to be asked and how the user decides to continue in each interaction with the system. Therefore we will have to consider creating other options *on the fly* by consulting the available knowledge sources.

In order to apply the domain knowledge sensibly we shall set out a few basic (informal) requirements that need to be met by a search system that utilizes the domain model in this sense. We want such a system to:

- allow the user to refine or relax the information request if necessary
- take as little as possible (in terms of dialogue steps)
- present the user a number of choices to continue the dialogue if this is appropriate
- present the best possible choices only
- avoid unnecessary dialogue steps.

To illustrate this, imagine a *Yellow Pages* lookup. An ordinary request would start off by selecting classifications followed by finding the appropriate advertisements listed under the classifications. This is the obvious strategy for a printed directory, and it seems useful to adopt that for an online system as well (be it a classified directory or a Web site).

Consider the somehow extreme example of a user searching for *"visa services"*. It seems logical to first select the relevant classifications from the classified directory and then if necessary ask the user to refine or relax the query. A less useful approach would retrieve all advertisements that can be matched against the two terms *visa* and *services* (a strategy that would deliver a large number of unrelated documents for this request since *visa* could simply refer to a payment method). True, a system like *Google* does record different types of matching terms depending on the context they were found in. However, distinguishing a number of different flat keyword tables does not differ *in principle* from the simple data-driven IR approach. The point is that in a search system that uses conceptual information encoded in a domain model, we try to separate relevant information from less relevant information at the model construction stage rather than at retrieval time.

Because it is a common problem that a query typically recalls *"too many"* matching documents one can easily construct numerous examples where the *hierarchy-driven* idea seems more appropriate than just ranking the results

retrieved by matching a user query against database entries. Consider a *Yellow Pages* request for *alarms*. Possibly relevant classifications as found in the Colchester (Essex, England) directory are:

- Alarm systems
- Burglar alarms & security systems
- Car alarms & security
- Fire alarms
- Gas alarms
- Intruder alarms.

With or without knowing the explicit classifications it is not clear for the system which documents will be most relevant. In this specific case a single dialogue step could establish which of those classifications the user is after. To make the point more general, even if there are no explicit classifications like the ones just listed, it is desirable to create them automatically and then utilize them in the search process.

In other words, standard information retrieval can be seen as a *data-driven* strategy of matching query terms to document indexes, but in contrast to that we use structures similar to classifications in a classified directory which will allow a *hierarchy-driven* search strategy starting at the top level of general classifications going down to the document level as depicted in the example in Fig. 1.8. Our "classifications" are concepts. We use the term *clusters* in the figure. A cluster in this context is a set of documents that matches a set of "classifications" (represented by concepts). The top clusters in the hierarchy could be representing single concept terms (e.g. *alarms*), clusters further down will represent a number of concept terms (e.g. *alarms* and *burglar_alarms*) that describe a smaller set of documents. This example gives only a motivation, in reality there will be a (possibly large) number of hierarchies.

1.4 Structure of the Book

This book consists of two main parts, the first one describes the theoretical framework and the second one is about practical applications that are based on this framework.

The first part is structured as follows:

- We will start with a review of related work (Chap. 2). We will first discuss literature that has to do with the indexing and domain model construction steps. We then look at the related work that is concerned with the search process.
- This is followed by a chapter that explains the framework of how to characterize and how to process *partially structured* data, i.e. all the offline data processing steps including the construction of a domain model extracted automatically from the documents' markup structure (Chap. 3).

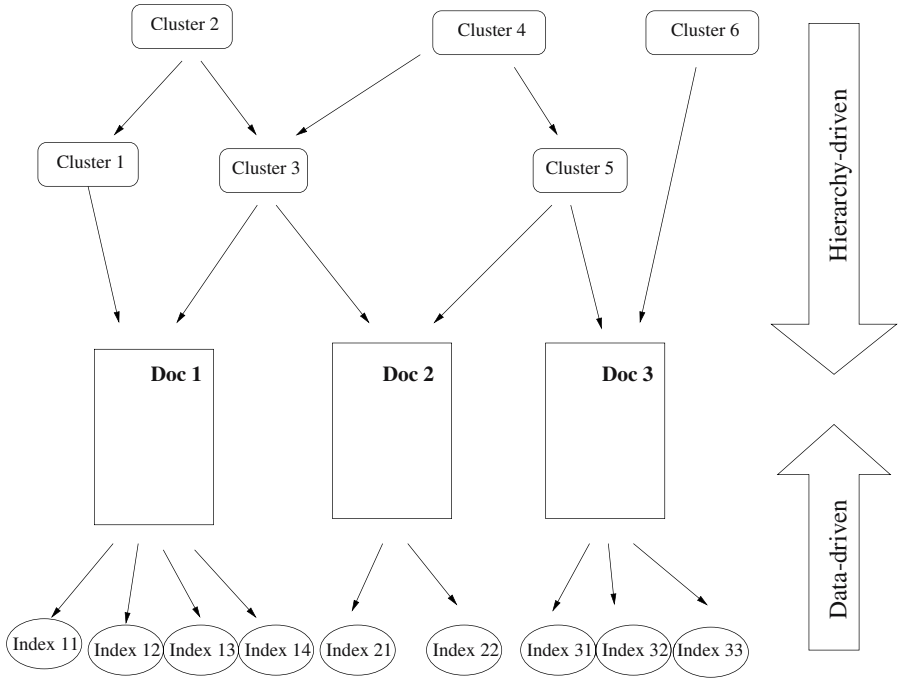

Fig. 1.8. Two search strategies

We will briefly outline how additional knowledge sources can be incorporated (Chap. 4).

- In the final chapter of the first part (Chap. 5) we will introduce the theoretical background of a specialized dialogue system that offers and makes choices about search through the set of documents based on the constructed index tables and domain models.

The second part describes some practical applications which employ those techniques and have been implemented as working prototypes:

- We will first introduce *UKSearch*, a search system for Web documents. Two domains will be discussed, the *University of Essex* Web site as well as the *BBC News* Web site (Chap. 6).
- The following chapter will discuss in detail a number of studies that we have performed as part of the development of the *UKSearch* system to demonstrate the usefulness of the outlined techniques. We will present evaluations as well as discussions of the results and conclusions that we can draw from these experiments (Chap. 7).
- We will then look at the YPA system, a search system for data in classified directories such as the *Yellow Pages* (Chap. 8).

We will conclude with an outlook on some future directions for the field (Chap. 9).

This book contains work published in earlier articles, namely [43, 49, 86, 87, 88, 89, 90, 91, 92, 161].

Part I

The Model

2

Related Work

It seems natural to break down the discussion of related work into two parts. We will first review the literature related to the data analysis and knowledge acquisition step (Sects. 2.1 - 2.7). We will then discuss approaches that exist to help users in the actual search process (Sects. 2.8 - 2.11).

There are a number of different communities doing research in the field of automatically processing, extracting and accessing partially structured data. Their approaches are quite distinct in some ways but overlap in other respects. The growth of the *Web* contributes to the fact that a combination of techniques is becoming more common. This "hybridisation" means, that some of the work discussed here would equally fit under different headings. Many of the approaches apply natural language processing or machine learning ideas, or both (which is why such headings have been avoided in this chapter). Roughly speaking, any approach to solve the problem of finding the right information for a user's need from a set of partially structured documents either relies on enormous amounts of data or on structural information associated with the documents, neither of which we can take for granted in the context of this book. The following is by no means a comprehensive overview of related work, but we do discuss the main issues with respect to the processing and interpretation of the data relevant in the context of this book.

Once we have indexed the data sources, we need to be able to access them quickly and efficiently, making use of all the structure and information we have detected in the indexing process. After reviewing the literature that is mainly concerned about the indexing process, we will then look at how the indexed data sources can be searched. More specifically, we will concentrate on search that goes beyond a simple match of a user query against a backend database but involves some sort of dialogue between user and system. This type of search can be an extremely simple interaction but it might as well be a fully fledged dialogue system. We will look at advantages and disadvantages of particular approaches before characterizing a simple dialogue system that seems most suitable for the type of data discussed in this book.

Again we look at a rather broad field of research. Research focus and objectives differ significantly between different research communities that work on dialogue systems. A large number of classifications of types of dialogues exists. One such classification would characterize our task as an *information seeking dialogue*. In the following we will concentrate on this particular type of dialogue and dialogue systems. Some of the most closely related work in this field is research into Web search and browsing as well as dialogue systems that have been developed in the speech community.

As before, the sections do not strictly follow the boundaries between research communities, nor is the discussed work exhaustive. The aim is however to highlight literature that is most relevant in this context.

2.1 Information Retrieval

Indexing documents has been a research topic in the information retrieval (*IR*) community for many years [133]. The traditional task is to select a set of documents from a large collection so that the selected documents satisfy a user's query. The Text Retrieval Conference (TREC) series has contributed to the fact that this research area now attracts a much wider interest than in the decades before.

To enhance *precision* and *recall* in an IR system there have been approaches to abandon the earlier idea of treating documents as a collection of keywords and take other aspects into consideration. Natural language processing (*NLP*) has been incorporated into IR systems (lexical processing, stemming, POS tagging etc.) [101, 140]. However, there is an ongoing discussion about how much linguistic knowledge can contribute to IR systems. Spärck Jones[1] argues that TREC has continued to cast doubt on the added value, for ad hoc topic searching, "on the value of sophisticated natural language processing for retrieval" [145]. Whatever the outcome of this debate, "NLP techniques used must be very efficient and robust, since the amount of text in the databases accessed is typically measured in gigabytes" [171].

The *Natural Language Information Retrieval* system *NLIR* [146] uses parallel indices which represent the same document using different indexing methods. This stream architecture contains index tables for word stems, phrases, proper names etc. which are merged to get a better overall result than applying each of the streams individually. Simple linguistically motivated indexing turned out to perform no better than statistic approaches, while more sophisticated ideas like concept extraction proved too expensive for large data collections [147].

Another idea is to extract the semantic content from documents in order to link related documents. Green proposes to use *lexical chaining* for that [59]. He focuses on the problem of synonymy and polysemy: documents concerning

[1] *We will spell the name Spärck Jones with an umlaut throughout the book because that is the spelling used on her homepage:* `http://www.cl.cam.ac.uk/users/ksj/`

the same topic do not necessarily use the same words. In a first step all chains of semantically related words are detected based on *WordNet*'s synsets and in a second step for each pair of documents this information has to be compared. In the reported work no significant improvement over standard IR techniques was noted.

We are interested in these developments because although the processing techniques presented in this book focus on *structural* rather than *linguistic* information, we are typically not dealing with enormous amounts of data that would prohibit the inclusion of some linguistic analysis. We do in fact apply low level linguistic processing (stemming, part-of-speech tagging and selection of phrases).

Finally, the emergence of the Web has triggered a growing interest in information retrieval techniques. A natural consequence was that a Web track was introduced in TREC. One question for the *Small Web Task* was to find out whether link information in Web data can be used to obtain more effective search rankings than can be obtained using page content alone. First investigations showed that hyperlinks did not result in any significant improvement [136]. The conclusion of these early results was that no measurable benefit was gained on standard TREC retrieval measures through use of links [63]. However, a case study did show that commercial Web search engines (that do use link structure) are significantly better than state-of-the-art TREC algorithms in finding Web pages of an entity such as a company, a university or an individual [139]. New TREC results give rise to similar conclusions. Results have shown that in order to find homepages (of a company for example) the exploitation of link structure was very effective. On the other hand, for standard ad hoc search exploitation of link information did not help [62]. The TREC conference recently distinguished a *topic distillation* and a *navigational* task in the Web track. The first one aimed at finding a set of homepages relevant for a query (this is different from *ad hoc* search where all relevant results are to be located). It was found that the use of anchor text was important, stemmimg was often helpful and URL information and link structure can also be helpful. The *navigational task* aimed at finding a single Web page (such as finding a particular homepage), and here it was found that different representations of the documents based on the document structure and anchor text seemed useful. Link structure gave mixed results, and stemming was not necessary for most participants [39].

The information retrieval community is very active, and the reader should consult TREC's homepage[2] or ACM's SIGIR homepage[3] for the most recent developments and results.

[2] `http://trec.nist.gov`

[3] `http:/www.acm.org/sigir/`

2.2 Information Extraction

Information extraction (IE) systems typically do not aim at capturing all of the source data, but at extracting specific types like names, dates etc. [36, 40]. This makes IE systems usually very domain-dependent and that distinguishes them in principle from the approach discussed in this book. A typical IE system has access to large electronic data sources to identify proper names, company names, place names etc. Moreover, all IE systems rely to a certain extend on manual analysis to identify patterns of interest in the first place.

SCISOR (System for Conceptual Information Summarization, Organization and Retrieval) [125], an early information extraction example, works in the constrained domain of corporate mergers and acquisitions to support imprecise and inexact queries as well as conceptual summarization by conceptually indexing the complete input documents. However, it makes heavy use of large conceptual databases.

In *InfoExtractor* [141] semistructured documents are split into various regions (*evaluation contexts*). In each region only certain concepts are searched for. The analysis identifies a set of predefined concepts in the documents. The authors rule out approaches based on inducing rules from the automatic analysis and classification of large numbers of documents because of a lack of a large training corpus [142].

One specific example of a supervised machine learning approach to learn IE extraction rules is *WHISK*. It learns rules in the form of regular expressions and differs from a number of other systems in that it has been developed to work on structured, semistructured as well as free text [143].

There are ways to minimize the manual effort by letting the system discover relevant patterns based on a set of *seed patterns* [168]. Another approach is to exploit highly reliable and easy-to-mine data sources - such as databases and digital libraries [35]. This allows a bootstrap approach which starts with some initially extracted information to obtain more complex modules such as wrappers (discussed further down). Those wrappers can then be used to collect more information that could again be the input for training more sophisticated IE engines. The key feature in the outlined approach is that it exploits the *redundancy* on the Web, i.e. the fact that the same information is typically available more than once and in a variety of forms.

There is a growing research interest in the process of capturing the semantic content of whole documents and representing it in an appropriate form. Essentially, this is an information extraction task. For semistructured documents this research was pushed by the introduction of XML [17]. The strength of XML lies in the ability to encode content rather than form. Curran and Wang for instance convert legacy data into XML [41]. HTML documents of a particular style are translated into XML by applying a transformation-based learning approach similar to that in the Brill part-of-speech tagger [18, 19]. Curran and Wang's learning algorithm is reported to need only a small training set. Another approach, which again requires a set of examples, is presented

by Zhang et al. [172]. Information extracted from HTML repositories is turned into structured records. A small set of examples is used to find occurrences of these seeds. These occurrences are grouped and patterns are recognized. These patterns are used to find new occurrences which can then be used to start the next cycle in this iterative process.

The database community too has developed a research interest in how to represent semistructured data, as well as how to extract parts of it. The *Lore* database management system [109] is designed specifically for the management of semistructured data. The *Information Manifold* system [100] allows uniform access to data in heterogeneous sources, by declaratively describing the contents and the query capabilities. A particular focus is the processing of *Web* documents, aimed at retaining structure and storing data in graph-structured data models by means of *wrappers* [94, 132]. This line of work is not just about accessing and retaining data but also about capturing structure, for example by transforming HTML documents into XML, or other formats. A recent example of a wrapper-based approach is *Building Finder* where information about buildings is retrieved from a number of different databases, e.g. geospacial data, white pages, property tax sites etc. [110]. However, any wrapper-based approach depends very much on a formally defined structure of the expected input or at least on a document structure that is fairly uniform across the collection. Our methods on the other hand do not rely on such assumptions.

2.3 Clustering

Regarding retrieval techniques, the idea of clustering documents into groups of related documents was pioneered by van Rijsbergen, who formulated the *Cluster Hypothesis* [155]:

> "Closely associated documents tend to be relevant to the same requests."

Much work in this field concentrates on the question of how to improve browsing through a document collection for a user who tries to find some particular information in a large pool of documents. Like information retrieval approaches, techniques for document clustering normally rely on statistics derived from huge amounts of data.

Scatter/Gather is a document browsing method for large information spaces that uses document clustering as its primitive operation [42]. It *scatters* the document collection into a small number of clusters. The user receives a short summary of each group of documents and selects a subset of clusters. The selected clusters are then *gathered* together to start the clustering process again with this subcollection of documents. This reduces the number of documents in the collection in each step until finally individual documents can be displayed. The system is designed for applications where it is difficult or

undesirable to formally specify queries. The clustering builds on words and word frequencies.

IBM's *TaxGen* text mining project aimed at the automatic generation of a taxonomy for a large corpus of unstructured news wire documents [113]. A hierarchical clustering algorithm first builds the bottom clusters and then works its way upward forming higher-level clusters by grouping together related clusters. The clustering is based purely on linguistic elements in the documents, e.g. co-occurrences of words (*lexical affinities*) or names of people, organizations, locations, domain terms and other significant words and phrases from the text (*linguistic features*).

An example of conceptually indexing a document collection (which can then be used for clustering) is *Keyphind* [61]. Machine learning techniques extract *keyphrases* from documents in the context of browsing digital libraries. This comes close to our idea of imposing a structure on the collection by extracting "important" phrases from each document, but in *Keyphind* the documents are much longer and furthermore a manually tagged training corpus is needed to build the classifier. *Extractor* is a similar system for extracting keyphrases using supervised learning [153, 154].

Clustering has also been used for *concept-based* relevance feedback for Web information retrieval [31]. Following a user query the retrieved documents are organised into conceptual groups. Note that this is significantly different from our approach: we aim at constructing a model for the entire document collection in an offline process, whereas Chang and Hsu use the search results to build their conceptual groups. One advantage is that we can use the model not just for ad hoc search, but we could as well assist a user who wants to browse the collection. Roussinov et al. discuss an empirical study that investigates how automatic clustering and user feedback can be combined in interactive Web search [129]. They use clustering to summarize retrieved documents and select terms that are related to the user query. The user has to decide which terms should be added. Their approach (called *adaptive search*) "utilizes Kohonen Self-Organizing maps and acts as a layer between the user and a commercial search engine". The study concludes that the system can suggest helpful terms and the approach increases the efficiency of information search.

Another example is *Grouper*, an interface to a meta search engine which dynamically groups the search results into clusters labeled by phrases extracted from the snippets that the search engines returned with the retrieved documents [169]. Online applications like this differ quite substantially from traditional *offline* clustering approaches where the whole document collection is clustered, rather than a number of retrieved documents. Post-retrieval document clustering has been shown to produce superior results. Moreover, clustering a few thousand documents is much quicker than doing this for millions of documents. However, for the work presented in this book a post-retrieval clustering approach is not what we want since we are interested in building a domain model for the *entire* document collection in advance.

There is a growing tendency to use clustering techniques in standard search engines to present the user with some query refinement options alongside the matching documents. Examples are *Teoma*[4], *AlltheWeb*[5], *AltaVista*[6] and *Vivísimo*[7]. However, an inherent problem of such techniques is to name the clusters, i.e. select the right text snippets to convey the *meaning* of the cluster or to select useful phrases that can serve as query refinement terms. For example, a search for *"Kruschwitz"* on the Teoma search engine recently returned a single refinement suggestion: *Freie Universit*, a piece of text that must have been derived from the (proper) German phrase *Freie Universität*.

A more advanced example is *Kartoo*[8], a meta search engine that evaluates the returned matches and generates maps that show the links between subsets of matching documents. The user can also refine or relax the query by choosing from a list of query modification terms.

Encouraging results for automatically creating names for clusters are reported by Glover et al. [58]. Clusters of Web pages can be accurately named by ranking words and phrases in citing documents. This is based on anchor text and text surrounding the anchors. The ranking of terms is based on *expected entropy loss* where the top ranked features by expected entropy loss are those which "occur in many positive examples, and few negative ones". Obviously, this requires training examples (something that we want to avoid). Existing *Yahoo!* categories were used as a benchmark.

The same problem, i.e. finding good phrases to describe clusters of documents in a set of matches returned by a search engine, is addressed in [170]. Potential phrases are extracted from the document titles and snippets and a learning algorithm is applied to assign a rank to each phrase. The algorithm exploits properties such as tf.idf, phrase length and phrase independence. It requires training examples since it is a supervised learning algorithm.

2.4 Classification

Document classification is closely related to clustering; the difference to clustering is that predefined categories exist, normally manually constructed, and after a training phase new documents can be classified on the fly. A major difference of standard classification approaches to our work is that we are not interested in *manually* constructing classifications, nor do we want general purpose classifications such as those found in Web directories like *Yahoo!* or the *Open Directory*.

A large number of complex manually constructed classifications schemes and taxonomies exist, ranging from general purpose classifications such as

[4] http://www.teoma.com
[5] http://www.alltheweb.com
[6] http://www.altavista.com
[7] http://www.vivisimo.com
[8] http://www.kartoo.com

the Universal Decimal Classification (UDC)[9] to domain-specific hierarchical structures like the Medical Subject Headings (MeSH)[10]. At a more abstract level specifications have been developed that can guide the construction of classifications and taxonomies. For example, the *Topic Map Paradigm* provides a "standardized notation for interchangeably representing information about the structure of information resources used to define topics, and the relationships between topics" [72].

What all these approaches have in common is that they represent knowledge with formally specified relations, which requires substantial manual encoding - very different to the bootstrapping idea that we have in mind.

Northernlight[11] is an example of a search engine where the results are grouped in dynamically created categories, or *Custom Search Folders*, which help a user narrow down the search in case of too many matches. The search used to cover the whole Internet and thus had access to much more data than what we assume. *Custom Search Folders* could not be offered if the query did not retrieve answers and needed to be relaxed, e.g. *"Minox dealer in Colchester"* or *"Minox Colchester"*.

Karlgren introduces the *Easify* interface that presents search results classified into different types of genres [77]. He shows that simple measures of stylistic variations in a set of retrieved documents can be used to distinguish document genres. He then argues that standard information retrieval systems cannot easily incorporate this type of information. "The aim is to utilize more knowledge about documents to improve user control over the information access process. Current systems have little knowledge of text, and interaction with them is designed after that fact. With more knowledge of documents, such as stylistic analysis can provide, the interaction can be richer." The outlined system combines classification based on user-defined, document-base oriented genres with dynamically created topical clusters.

Automatic classification of Web pages is the focus of the work by Attardi et al. [10]. They introduce *categorization by context* which exploits information surrounding the links pointing to a document in order to classify that document. It is seen in contrast to *categorization by content* which relies on textual information found in a document. The basic assumptions are: (1) a Web page which refers to a document must contain enough hints about its content to induce someone to read it, and (2) such hints are sufficient to classify the document refered to. Hints are mainly anchors, but also include structural information found in HTML documents like headings, titles etc. However, building the profiles of the classifications are left as open issues, and the two options presented are to either build them by hand or use some learning techniques.

One problem with classifying Web documents as opposed to more traditional classification tasks is the heterogeneous nature of HTML documents.

[9] `http://www.udcc.org`

[10] `http://www.nlm.nih.gov/mesh/meshhome.html`

[11] `http://www.northernlight.com`

An interesting approach is to create *megadocuments*, the concatenation of all documents in the training set that have been classified in the same category [81]. A new document gets assigned to the category whose corresponding *megadocument* is most similar to it.

A step further is hierarchical classification, e.g. [46]. Web pages (such as pages returned by a search engine) are classified into a given hierarchical structure of categories. Unlike other approaches, Dumais and Chen use a large collection of very heterogeneous web content, and the classification is based on support vector machines (SVM). The classification problem is decomposed into a set of smaller problems by utilizing the hierarchical structure. The actual classification uses flat summaries of the documents rather than the complete Web pages. Title, keywords and some description extracted from the meta tags or the first 40 keywords of the body are used to represent the documents. In a different paper they present a user study discussing HCI issues [32]. The organisation of Web search results is the focus, and they conclude that a category interface is superior to a ranked list of documents.

Existing classification structures have also been employed to help the user in the search process by matching the initial user query against a classification hierarchy and displaying those parts of the hierarchy which appear to be most relevant to the query. The relevance can be calculated based on a vector representation of each classification that has been pre-computed representing the centroid of all documents and subcategories associated with this classification. This is the approach taken by the *ARCH* system [118]. However, such a system requires a classification hierarchy (with corresponding documents) in the first place.

Another idea is to use an existing classification structure (taxonomy) with all its classified documents, such as the *Open Directory*, as a source for extracting query modification terms that make the original query more specific. *TACC* (taxonomy-based context conveyance) is an example where the user needs to select a particular category first. The user query and the selected category are then used to construct a modified (more specific) query that gets submitted to a search engine [117].

2.5 Web Search Techniques

Web pages contain more structure than many other document collections. In a typical HTML page, for example, one can find some text marked up as the document title. Furthermore, one would typically find hyperlinks to other documents which would include anchor text to describe these documents. Many other tags can be used to highlight parts of the text or distinguish parts of the document from the rest. All this structure can be used to extract the semantic content or to judge the relevance of a specific page. The structure that is exploited in this context is internal document markup or hyperlinks between documents, or both. One of the reasons to exploit such structure is the

observation that Web pages are not always best described by their content. Kleinberg reports that companies like *Yahoo!*, *Excite* and *AltaVista* do not use the term "search engine" on their pages. Similarly, the term "automobile manufacturers" cannot be found on the Web pages of Honda or Toyota [82]. Hence, looking at the link structure might result in a better picture of what a Web page is about. We will look at some of the most prominent approaches. Note that we do not use any link structure by default in our knowledge extraction techniques. To extract conceptual information we simply exploit the markup structure found in a document and no references to the document from elsewhere. However, hyperlink information can be incorporated. Furthermore, the work discussed in this section is very much related to the book because the underlying motivation is similar to our problem: to find the most relevant documents for a user query in a collection of Web pages.

Amitay proposes to look at anchor texts of pages that point to a document and find similarities between pointing anchors [6]. The *Hyperlink Vector Voting* approach applies similar ideas [102]. Rather than depending on the words appearing in the documents themselves it uses the content of hyperlinks to a document to rank its relevance to the query terms. This overcomes the problem of spamming within Web pages and seems appropriate for Internet wide search but would cause problems in subdomains where the number of links between documents is much smaller, and certainly problems will occur for those pages which are refered to by a small number of documents only, or no documents at all. One can go further by employing not just the anchor text but also additional structural information found in the context of the hyperlink as suggested by Fürnkranz [55]. For the task of classifying pages using a given set of classes Fürnkranz reports that it is possible to classify documents more reliably with information originating from pages that point to the document than with features that are derived from the document text itself. Results reported by Glover et al. [58] are consistent with this: Web pages can be classified significantly better when using "extended anchor text" (i.e. anchor text as well as text surrounding the anchor) instead of the text found in the actual document.

Anchor text can also be exploited to construct potential query refinement terms which are significantly better than those derived from the document content [85].

HyPursuit is a hierarchical search engine that clusters the documents based on both document contents and hyperlink structure [163]. The content-link clustering algorithm applies a document similarity function that is based on both term similarity and hyperlink similarity factors. The measure of hyperlink similarity between two documents captures the path between two documents, the number of ancestors and the number of descendents that both documents refer to. Term similarity between two documents is based mainly on term frequency with heavier weight assigned to terms with attributes *title, header, keyword* and *address*.

The *Clever Project* [29] looks at topic related search. The domain is the complete Internet and the problem to be solved is filtering out those pages which are truly relevant for a specific topic, i.e. the problem of too many matches. *Authorities* and *hubs* are distinguished, places that are either relevant or are collections of links to those pages, respectively. *Authorities* and *hubs* are found by purely analysing the connections between Web pages. This is based on the HITS algorithm developed by Kleinberg [82]. A modification of that algorithm was presented by Chakrabarti et al. [30], again as part of the *Clever Project*. The extraction of *hubs* and *authorities* is performed by a combination of text and link analysis. The text in a window around the hyperlinks is evaluated and the initial weight of a hyperlink which is set before the iterative calculation starts is increased with the amount of topic-related text. A related approach is discussed by Modha and Spangler [111].

The *Cha-Cha* system has been developed for *intranets*. It imposes an organisation on search results by recording the shortest paths to the root node in terms of hyperlinks [33]. But this is only applied once results could be found by using the search engine. It exploits hyperlinks but ignores the internal structure of the indexed Web pages.

However, internal structure of Web documents is being incorporated more and more in standard Web search engines. A prominent example is *Google* whose development was inspired by the idea of improving the quality of search as opposed to efficiency [20]. It makes use of both link structure and anchor text. Each page gets a ranking value depending on how many other pages reference to it. Moreover the system associates Web pages not just with the text in it but also with the text found in anchors linking to this page from elsewhere. This makes it possible to return pages that have not even been crawled. Furthermore, *Google* keeps track of some visual presentation details like font size to give for example heavier weight to words which appear in a larger font than the rest of the document.

The question is how much markup structure and, more specifically, which particular markup tags can easily be utilized to improve the search process. In other words, which markup tags should we consider when trying to extract knowledge from documents automatically? A number of options should be explored here.

First of all, solely relying on particular markup contexts and ignoring all other text in the documents seems not a very good idea. Although the index size can be reduced significantly, it has been shown that indexing only the title, anchor and emphasized text leads to very poor results in terms of precision and recall [71].

Secondly, it has also been shown that link text is a useful indicator for the semantic content of a Web page [70]. According to this study, 61% of the appropriate text was judged helpful or very helpful in indicating the page content. However, the same study concluded that heading text is much less useful.

Thirdly, Pierre reports that only a third of all Web documents investigated in a classification task contained meta tags. Nevertheless, if the documents contained meta tags, then they could be classified most accurately entirely based on those tags [121].

Finally, for search in intranets anchor text has been found to be highly effective, document titles however were considered far less useful [47].

The conclusions we draw from these observations are:

- A sensible approach to construct a domain model automatically (exploiting the markup structure of documents) could be based on a limited number of commonly used markup tags.
- Such a domain model cannot be a substitute for a comprehensive document index. A search application that employs the model should not use it as the *only* knowledge sources, but combine it with a search engine that has access to the full index database.

2.6 Ontologies

Significant work has been going on in recent years in the *knowledge representation* community. Much of that work is closely linked to the idea of building a *Semantic Web*. This research area is not simply about representing information but covers a whole range of related issues. For example, an entire issue of *IEEE Intelligent Systems* was dedicated to ontologies reporting on work such as learning ontologies [106], a proposal for a standardized language to express and represent ontologies called OIL (now superseded by the *Web Ontology Language* OWL, a recommendation of the World Wide Web Consortium (W3C)) [51], as well as tools and techniques for generating and processing semantically enriched content [64]. All this work is quite different to the idea presented in this book, in that we apply a *bootstrapping* approach to build some representation of a document collection which only reflects what can be derived from the actual data. The result is not an ontology but a set of simple hierarchies in which the type of links between nodes is not formally specified. Nevertheless, ontologies are a very active (related) field and some work will be discussed here.

Ontologies and customised versions of existing language resources like *WordNet* are being successfully employed to search product catalogues and other document collections held in relational databases [60, 53]. Alani et al. introduce the *Artequakt* project [4]. In this project ontologies are used to guide the extraction of knowledge from Web documents, more specifically bibliographic knowledge about artists. The system uses an ontology in combination with *WordNet*, inspired by the observation that traditional information extraction systems "lack the domain knowledge required to pick out relationships between the extracted entities." However, a domain-specific ontology is needed in the first place.

Another example is *CS AKTive Space (CAS)* that allows a user to explore information about computer science research in the UK [137]. But the problems that developers of systems like *CAS* face are similar, namely the difficulty to encode the documents in some appropriate format and to construct knowledge sources such as ontologies that can then be used to reason about the data available.

An example of using ontologies to build a *structured* index of a Web site is presented by Desmontils and Jacquin [45], where the indexing process consists of a number of steps, including the construction of a flat index (where the weight of an index term is based on frequency and HTML markup contexts), the selection of concepts and the mapping of documents against the ontology. Again, this process assumes that an ontology exists.

Part of the research in this field is the actual construction of ontologies and large knowledge bases (or in fact the mapping from one knowledge source to another one, e.g. [56]). The cost to create the resources can be enormous and it is difficult to apply these solutions to other domains where the document structure or domain coverage is not known in advance. There is also an ongoing discussion as to how applicable ontologies are. Spärck Jones argues that the Text Retrieval Conference series (TREC) "has continued to cast doubt on the added value, for ad hoc topic searching, of structured classifications and thesauri or (to use the currently fashionable term) ontologies" [145]. This is part of the argument referenced earlier in the section on information retrieval (Sect. 2.1).

There are some fundamental problems with ontologies. A major problem is that "there is a lack of methods and tools supporting and facilitating ontology reuse"[104]. This is a motivation for Maedche et al. to introduce an infrastructure to register ontologies, to allow the reuse and to support the evolution of ontologies.

There is some more related work that should briefly be discussed here where the focus is more on the automatic knowledge acquisition side.

Moldovan et al. acquire domain-specific knowledge by using *WordNet* as a core ontological knowledge base and enrich it with relations found in a corpus of documents [112]. It is necessary to identify *seed* concepts to start with. The knowledge extraction is based purely on linguistic information, and new concepts and relations are derived from isolated sentences. A human inspects the results and accepts or rejects them.

Fujii and Ishikawa describe their work as either linguistic knowledge extraction or (from an IR point of view) as construction of domain-specific Web search engines [54]. Their methods extract knowledge from HTML documents by looking for patterns that are typically used to describe terms. These patterns can be language-based or markup-based. However, the methods actually aim at finding *term descriptions*. Furthermore, the NLP-based extraction methods are language dependent, while the HTML-based methods are essentially investigating particular tags and the text immediately following those tags.

2.7 Layout Analysis

By *layout analysis* we mean the process of uncovering the structure of a document based on layout cues. Most work in this field relies on documents whose format is known, very different to what we assume. The generic approaches we present here are more relevant in the context of partially structured documents with unknown structure, and hence to our purpose.

One motivation to layout analysis is to convert printed collections of documents into their corresponding tagged electronic version. *Autotag* [152] is an early layout analysis system that aims at processing many types of documents based mainly on heuristic rules. It performs a physical followed by a logical analysis. The intention is "to capture every conceivable object that may add to a document's presentation". However, it does not go as far as to capture the semantic content of the documents.

Finding the logical structure of a document without knowing the actual documents or styles used in them is the focus of Summers' work [150]. Her approach follows a two-step process: *segmentation* and *classification*. Segmentation primarily uses layout, contour and font shape information. The classification step compares the text segments with predefined structure prototypes, mainly by means of geometric cues. The only linguistic knowledge used in the described prototypes is the identification of typically appearing or non-appearing symbols. If some information about the *style* of the documents is known, then this can be incorporated in the processing steps as well.

Layout analysis is another research area where the development of the Web triggered a demand for more sophisticated methods to understand the structure of a document. One motivation is to find out how important individual segments of a Web page are. This is the starting point for Song et al. for example [144]. Their algorithm first segments a Web page using HTML tags and layout features such as font, colour and size. In a second step these segments are arranged according to their importance. Machine learning techniques are used to train the algorithm.

Other layout analysis work is much more specific about the type of structures to be detected and extracted, and resembles more an information extraction task than anything else. However, there seems to be a lot of potential in exploiting more layout structure. Henzinger et al. believe "that the exploitation of layout information can lead to direct and dramatic improvement in web search results" [65].

2.8 Web Search Studies

Now that we have reviewed the literature that focuses on the data analysis and acquisition process (and compared it to our methodology) we can move on to discuss approaches for assisting users in the actual search process.

Web search as such is obviously closely related to the overall focus of this book. Some Web search techniques have been discussed in the context of indexing the data sources and acquiring knowledge from those sources. These processes do not usually involve the user but are often performed offline before the user can actually start searching the document collection. In this section we are more interested in all those aspects that are important for the actual interaction between user and system although it is sometimes difficult to separate these issues.

We shall first look at some motivating investigations concerning the users' behaviour when searching the Web. The most comprehensive study of Web queries so far was conducted by Silverstein et al. [138]. An evaluation of a large log file of nearly a billion queries submitted to the *AltaVista* search engine in a period of 43 days comes to a number of interesting conclusions. First of all, queries are normally very short. The average length of a user query is 2.35 words (similar results are reported in an earlier but smaller evaluation of queries submitted to *Excite* [73] and in a more recent study of queries submitted to *America Online* [14]). That means queries on the Web are shorter than in more traditional information retrieval systems. Secondly, the 25 most common queries account for 1.5% of all queries, even though they are only a small fraction of all unique queries. The query *"sex"* is by far the most frequent one. As the third interesting result it was found that "surprisingly, for 85% of the queries only the first result screen is viewed, and 77% of the sessions only contain 1 query, i.e. the queries were not modified in these sessions".

There are at least two lessons to be learnt from this work. First of all, user queries are generally very short which will naturally lead to a large number of documents being returned. But the second aspect is that the majority of users did not perform any query modifications. A system which applies a model of the domain to propose possible query refinements must perform extremely well to be accepted by the user.

Furthermore, a number of studies have been conducted to find out whether the search process could benefit from offering potentially relevant terms to the user in an interactive query expansion process (this is different from automatic query expansion where the user does not get directly involved in the selection of query modification terms, e.g. [123]). In one of these studies by Margennis and van Rijsbergen potential expansion terms are automatically derived from the documents retrieved by the original query [107]. The underlying assumption reads as follows:

> "It seems reasonable to assume that a searcher, given a list of the query expansion terms, will be able to distinguish the good terms from the bad terms."

It was found that interactive query expansion performed by an experienced user has a potential to significantly improve the search process. However, they also found that inexperienced users did not make good term selections and

hence interactive query expansion led to no improvement in the search process. Margennis and van Rijsbergen conclude:

> "Without good strategies and careful reasoning it is unlikely that a searcher will be able to use techniques such as interactive query expansion effectively".

Interestingly, Anick found that even if the search engine presents the user with query modification suggestions (in this case via *AltaVista's Prisma* tool), then the vast majority of reformulations are still done manually [7]. However, he also found evidence that a subset of those users presented with query modification terms did make effective use of it on a continuing basis.

Query modification suggestions are by no means restricted to text-based retrieval systems. Image retrieval, for example, may also benefit from presenting representations of images returned from an initial query to help refining the search as suggested by the *Blobworld* framework [25]. However, we will not elaborate on this since our research interest is primarily in text documents.

An interesting evaluation of different types of Web search has shown that some guidance to help the user in reformulating a query can significantly improve the relevance of the retrieved documents compared to standard Web search [22]. Part of the study was the comparison of Web search using *Google* and a search via the *Hyperindex Browser*, an interactive tool which passes the user query to a search engine and displays a list of linguistic phrases found in the top matching documents returned by the search engine. Those phrases can be selected by the user to constrain the query.

Recent experiments that have been conducted as part of the TREC interactive track have shown that hierarchical clustering and summarization can significantly help the user to locate relevant documents quickly. The experimental search system called *HuddleSearch* uses a hierarchical clustering algorithm which dynamically organises a set of retrieved documents for a user query. Uncompleted tasks and average time to finish a search task were reduced by the use of *HuddleSearch* compared to a standard list-based search engine [116].

Finally, we want to point out that most research into hypertext search technologies concentrates on internet (Web) search, but very little has been reported on search in intranets. For a number of reasons intranet search is very different from internet search. Standard ranking functions such as PageRank and HITS that work well for Web collections tend to be less effective for intranets [48]. Furthermore, one can expect very little spam in intranets compared to the Web in general.

2.9 Navigating Concept Hierarchies

As discussed earlier, one can tackle the problem of matching the user's information need against the available data by processing the data into some

conceptual model that can be used as a tool for guiding the user to a small set of relevant matches. The model can either be constructed offline or online by investigating the documents retrieved for the initial query. The system can then apply the model to help the user navigate through the answer set. There are at least two good reasons for this. First, the user gets a feel for what data is actually available. Rather than guessing new query terms, the system presents the query in context and proposes possible query refinement or relaxation terms. A second reason is to support a user who wants to browse rather than search a document collection, because "current search mechanisms are not much use if you are not looking for a specific piece of information, but are generally exploring the collection." [119].

Research on the construction of concept (or term) hierarchies has so far focussed mainly on word co-occurrences or linguistic information. None of the approaches we will discuss uses the actual document markup structure.

The motivation for the work by Sanderson and Croft [134] is to automatically organize documents in a conceptual structure based on keywords found in the documents. Unlike in manually constructed subject hierarchies, this structure is customized to the set of documents. The presented solution is a hierarchical organization of concepts automatically derived from a set of retrieved documents. General concepts are placed in the higher levels of the hierarchies, more specific ones further down. The construction is based on pairs of terms occurring in the same document, independent of the context they were found in. A concept x is considered more general than a concept y, if the documents which y occurs in are a subset of documents containing x. Experiments have been conducted comparing the use of these hierarchies with simple list interfaces in *interactive query expansion* [74]. The results show that users who applied the hierarchical representation completed the expansion task in a significantly shorter time than users who were given a list of expansion terms. However, no significant differences in terms of precision and recall were found.

Closely related is the *Paraphrase Search Assistant* [9]. The focus is again on how to help the user find the set of most relevant documents among a large set of documents retrieved for a query. The user is presented a list of relevant terms, each of which expands to a list of compounds (containing the selected term) which can then be used to constrain the query. The assumption is the "tendency for key domain concepts within result sets to participate in families of semantically related lexical compounds". An interesting aspect in this context is that query logs have shown that users actually make use of the presented terms for query refinement. A more detailed account of this work can be found in Anick's PhD thesis [8], where two systems are presented. One of them, *Paraphrase I*, identifies a set of topics (*facets*) by clustering the document collection prior to query time. The second system, *Paraphrase II*, constructs the *facets* on the fly. In both cases the facets are then used to expand them into sets of compounds.

The automatic construction of hierarchical information utilizing the distribution of *features* is discussed by Glover et al. [57]. Features are words or phrases of length two or three. Based on a small set of sample Web pages, the method finds *parent*, *self* and *child* features, where *self* terms are good descriptors for a selected cluster of documents (e.g. "biology"), *parent* terms describe more general (e.g. "science") and *child* terms more specific concepts (e.g. "botany"). Note that this method needs to have a set of positive examples in the first place. Based on that one can infer the structure by comparing the distribution of terms in the positive examples with the distribution in the entire collection.

A simple algorithm to construct graph models of related words is introduced by Widdows and Dorow [167]. The method can be used for "assembling semantic knowledge for any domain or application" and is based on grammatical relationships such as co-occurrence of nouns or noun phrases and needs only a corpus tagged for part-of-speech. The underlying motivation is similar to what this book is about: to uncover semantic relations by looking at the actual data, in other words "to observe word meanings with no prior agenda: to hear the corpus speak with its own voice" [166].

Another linguistically motivated approach has been suggested by Paynter et al. [119]. A phrase hierarchy is automatically constructed to facilitate browsing of a Web site. Noun phrases are extracted from the document collection and organized in hierarchies of phrases. The further down one goes in these hierarchies, the longer the phrases are. Hence, a user can start exploring the collection by searching for *forest*, then select *forest products*, from there go to *wood forest products* etc.

A technical evaluation of different techniques for automatically building concept hierarchies was performed by Lawrie and Croft [98]. Basically, the work compares the *lexical modification* approach [9] and the *subsumption* technique [134] to build hierarchies that can be used for browsing document collections. The findings are that the strength of the subsumption approach is to separate documents into small groups; lexical hierarchies on the other hand do a better job of including all documents in the hierarchy. Another interesting aspect of the presented results is that clustering was found to improve the results. The original document set is clustered, and hierarchies are built for each cluster separately. The resulting concept hierarchies contain more relations than single hierarchies and have a shorter average path length. The method of clustering was found to have little effect on the quality of the resulting hierarchies.

Finally, there has been work on the "description of a linguistically motivated method for identifying and browsing index terms and establishment of fundamental criteria for measuring the usability of terms in phrase browse applications" [157]. Documents are indexed by a system called *LinkIT* which parses the text, identifies noun phrases and ranks them according to their frequency. The assumption is that the "distinction between phrasal heads (the most important words in a coherent term) and modifiers serves as a basis for

the hierarchical organization of terms." It was found that the vast majority of heads have two or fewer different possible expansions. For those heads a hierarchical index would be created and displayed on user request. The conclusion is that the linguistically motivated structure of index terms helps the user to efficiently browse and disambiguate terms.

2.10 Dialogue Systems

Dialogue systems are a research area that attracts a number of quite different communities. Since we are interested in some fairly specific type of dialogue systems, namely those that can be summarized as *information seeking* systems, we will concentrate on related work in this area. That means that we will not discuss systems such as *Verbmobil* [158], a speech-to-speech translation system where the dialogue system mediates between two human users. Nor will we discuss dialogue approaches that try to model psychologically motivated ideas. We will focus on practical approaches involving systems that assist a user in accomplishing some search task in the widest sense.

The DARPA *Communicator* project aims at rapid development of multimodal speech-enabled dialogue systems [159]. It follows a schema-based approach. Schemas describe major information units. The domain is travel itineraries. This is very different from just searching for documents. The two main activities in creating an itinerary are the composition of an itinerary structure and the population of this structure. One of a number of example implementations is CMU's *Communicator* system [130]. Rather than following a strictly system initiated dialogue it uses a mixed initiative approach giving the user more freedom. This decision was based on the experience "that users generally dislike the rigid policy". An agenda keeps track of what tasks still need to be fulfilled (i.e. what information needs to be collected from the user). The complex data structure, the *product*, that needs to be filled with information, has handlers attached to it that encapsulate knowledge necessary for interacting about a specific information slot [162].

The EU funded *Task Oriented Instructional Dialogue (TRINDI)* project addressed central issues in designing dialogue systems that could be used for interaction between humans and computers. Key aspects of this research concern the representation of information states and information about how the system would move between dialogue states. A toolkit (*TrindiKit*) has been constructed that serves as a framework for the experimentation with different theories and applications [95]. However, we will introduce a much simpler representation of dialogue since that seems to be sufficient for applications such as ad hoc search.

A large number of dialogue systems is concerned with time and schedule information, such as OVIS [115], the Philips train timetable system [11], Sundial [120, 108], TRAINS [5]. These are heavily restricted domains, in terms of size and complexity of data. In addition, there are some systems which

retrieve addresses from *Yellow Pages*, e.g. *Voyager* (and its sibling, *Galaxy*, which the *Communicator* framework is based on) [174, 173].

HappyAssistant is a natural language navigation system that helps a user find relevant information about products and services on an e-commerce Web site [27]. A dialogue is initiated by matching concepts from the user query to "business rules". The system initiated interactions allow to collect more information about the user needs until the system can recommend an item or until all possible items have been eliminated. This system served as a prototype for the *Natural Language Assistant* (NLA) [26]. However, both these systems access a well structured product database.

All the above-mentioned systems involve the filling of some information slots and then reacting appropriately to the set of filled slots.

Finally, information assistants have also been built mainly based on machine learning ideas. As mentioned before, the machine learning community has an interest in understanding documents to automate the extraction of information from documents like Web pages by means of wrappers. Wrappers can be used to extract information in real time. Examples of information assistants based on this idea are the *Travel Assistant* and the *WorldInfo Assistant* which interact with a user by collecting values to fill slots [83, 84]. Groups of slots are organized in *templates* and arranged hierarchically into *templates* and *subtemplates*. Constraints define how particular types of information are related. The actual results (i.e. travel itineraries or geographical information about particular regions) are collected from a variety of Web pages using wrappers. This is very different to what is described in this book. The emphasis is much more on the planning part.

2.11 Usability Issues

It is worth reviewing some more work on usability issues in the widest sense. A number of studies (ranging from preliminary investigations to some larger scale studies) have come to conclusions that are relevant here.

The work on the *HappyAssistant* dialogue system included user studies which showed that the number of clicks as well as the time spent on searching could be reduced significantly by replacing a menu driven system with natural language navigation [27]. Interestingly, "the vast majority of user queries (85%) were relatively short, and consisted of noun phrases or lists of keywords". The conclusion is that "in such contexts, sophisticated dialog management is more important than the ability to handle complex natural language sentences". Studies have also shown that the users clearly prefered dialogue-based to non-dialogue-based searches [26].

A comparison of interactive search in a homogeneous domain using a keyword-based search engine and one that uses an ontology is reported by Sutcliffe and White [151]. For the ontology-based system the documents were attached to the appropriate nodes in the ontology. The search started at the

top level, proceeding step by step to the levels beneath. The application domain was word processing using the Lotus Ami Pro program. The result of the study was that the keyword-based system performed better than the one using the ontology. However, more interesting are the conclusions drawn from the study. One of them was that "if an ontology search does not lead directly to a correct answer it is essentially a complete failure", unlike keyword-based search. The characteristics that were identified to be important when using some taxonomy to search were:

- It must be possible to decide easily which category at a level is relevant to the query;
- Categories should ideally be mutually exclusive;
- The level of branching at any level must be limited;
- The number of documents attached to leaf nodes must be small.

Observational experiments conducted in the TREC interactive track focused on finding out whether users choose a particular presentation type for a particular search task [38]. The different types of presentations were ranked document lists, a clustering interface and an integrated interface combining ranked lists, clusters and expert links. The tasks were searches for single documents or sets of documents, respectively. It was concluded that users did not select a particular interface for a particular search task nor was any of the presentation types found to be superior for any particular task.

One may of course address the question of how a search engine should interact with the user very differently. Understanding the underlying information need of the user may lead to different system architectures for different tasks. Broder distinguishes three different classes of Web queries: *navigational* (aimed at finding a particular site), *informational* (to find information about a topic), and *transactional* (aimed at web-mediated activity such as online shopping) [21]. A study of queries submitted to *AltaVista* revealed that less than half of all queries are in fact informational queries. Another study (also investigating *AltaVista* queries) found that about 60% of user queries were informational and only 15% navigational, in other words "so-called navigational queries appear to be much less prevalent than generally believed" [128]. Such findings will influence the design of future search engines and they are also closely related to human computer interaction issues. The large field of human computer interaction shall however not be discussed in detail here. A comprehensive overview of related work and the problems involved can be found in the chapter *User Interfaces and Visualization* in the excellent text book *Modern Information Retrieval* by Baeza-Yates and Ribeiro-Neto [12].

2.12 Concluding Remarks on Related Work

To conclude this chapter, we found that techniques for the automatic extraction of knowledge typically rely on particular assumptions about the structure,

the language or the type of documents in the collection. There has so far been no attempt to simply use the markup structure to construct a domain model for a document collection. To do this we will need to make minimal assumptions about the documents in the collection. Our approach does however share ideas with some of the related work presented here. Furthermore, the use of a simple dialogue system to assist a user in ad hoc search appears to have the potential of helping a user to find documents more easily.

3

Data Analysis and Domain Model Construction

This chapter will address the data analysis and model construction issues in more detail. We will define *concepts* and relations between them. We will then discuss how the selected *concepts* can be arranged to obtain a model that is comparable to a classification hierarchy. The result will be our *domain model*.

3.1 Documents

Our primary aim is to assist users in the search process. So, we assume there is a collection of items which is mainly based on natural language text but could as well contain other elements like graphics or hyperlinks. We will call those items *documents*. Documents could be raw text, marked up text, articles etc. This loose characterization of a document allows us to treat advertisements in a classified directory as documents while in a different domain documents might be Web pages or news stories. We will also assume that a document contains words. These words are sequences of non-white characters separated by white space, i.e. words do not have to be lexical entries. We will refer to a *subdocument* as any coherent part of a document, be it words, phrases or whole paragraphs. From now on we will write $doc(d)$ as a short form to express that d is a document in our collection. If not otherwise specified, then this collection will be refered to as D. In other words the definitions in this chapter are to be applied relative to a fixed set D of documents.

The data sources of interest are *partially structured*, that means the documents do have some structure as illustrated in Chap. 1. We talk about data sources here because in addition to the documents there might be some explicit structure already present within the documents. That could be the organisation of documents in a file structure or some explicit classification of documents, both of which are not necessarily part of the documents themselves.

The document structure can come in a variety of forms; we are interested in what can be abstracted as some sort of *markup context*. Informally speak-

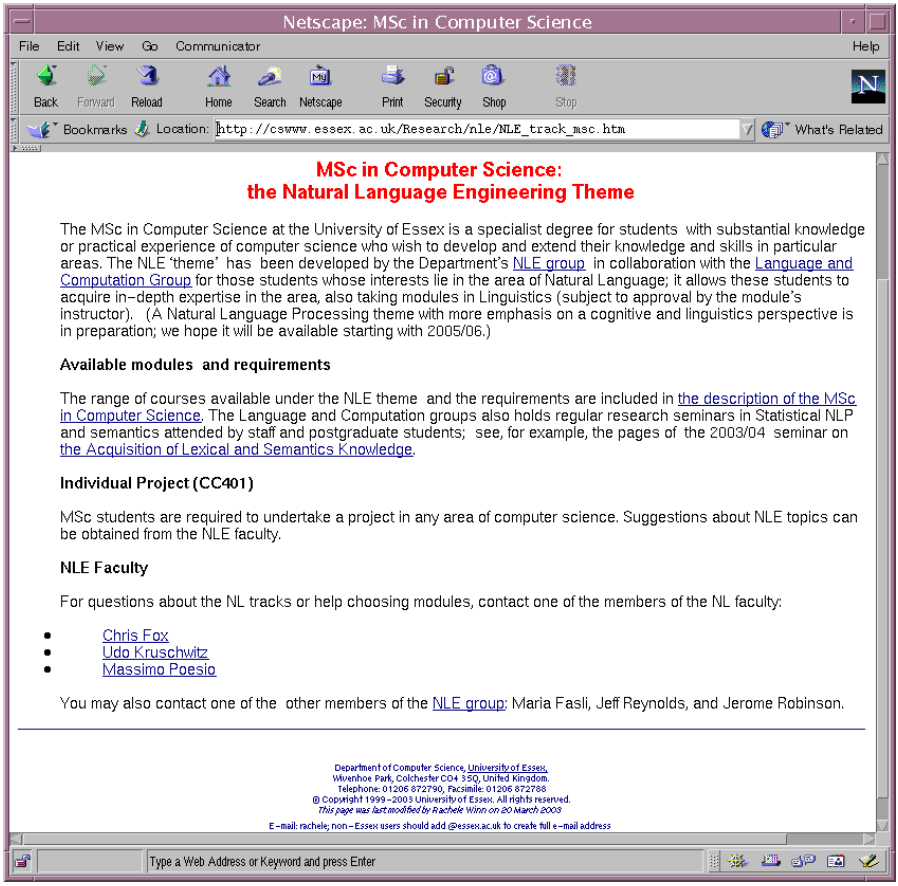

Fig. 3.1. Example Web page

ing, this is an explicit convention of highlighting or marking up a part of a document in a particular way distinguishing it from other parts. For example, in this book a chapter or a section title can easily be identified by the font size and type of the text (i.e. their markup context).

We write $markup(m, d)$ to express that m is a markup context for document d. We also define $markup(w, m, d)$ to express that a word (or a phrase) w in document d is found in markup context m. Furthermore, we will write $\{m|markup(w, m, d)\}$ to describe the set of markup contexts word w occurs in (in document d).

For hypertext documents such as the one displayed in Fig. 3.1 the different markup contexts to be considered would be defined by various kinds of tags that are used to mark parts of the text as heading, title, anchor text, normal text etc. whereas a collection of newspaper articles might distinguish font size,

shape, possibly a number of colours as markup contexts. This also implies a particular part in the document can be part of different markup contexts at the same time. For example, in this section of the book we might distinguish three markup contexts: the default font, bold and italic types.

Suppose, there is an indexing process, that is a function on subdocuments of document d to a set of index terms. Then we will write $index(t, d)$ to express that t was selected as an index term (or simply index) for document d in this process.

A standard IR indexing technique might select all content-bearing words and phrases and write stemmed word forms as an index into a database. In the applications that we will discuss later on we will index nouns and certain noun phrases.

Throughout the book we represent index terms consisting of more than one word as words connected by an underscore, e.g. *trade_union* or *department_of_industry*.

3.2 Concepts

Definition 3.1. *An index term c is a concept term (or simply concept) of type n for document d, written as $concept_n(c, d)$, if c occurs in at least n different markup contexts in d, i.e.:*

$$concept_n(c, d) \Leftrightarrow \big(index(c, d) \wedge \big|\{m|markup(c, m, d)\}\big| \geq n\big) \tag{3.1}$$

In addition, we define:

$$concept(c, d) \Leftrightarrow (concept_n(c, d) \wedge n \geq 2) \tag{3.2}$$

$$concept_n(c) \Leftrightarrow \big(\exists d\ index(c, d) \wedge \big|\{m|markup(c, m, d)\}\big| \geq n\big) \tag{3.3}$$

We will say that c is a concept of type n (or type-n concept) when we refer to the definition above (i.e. $concept_n(c)$). Analogously, we will refer to c as being a concept if we want to express the fact specified by the following definition (i.e. $concept(c)$):

$$concept(c) \Leftrightarrow \big(\exists d\ index(c, d) \wedge \big|\{m|markup(c, m, d)\}\big| \geq 2\big) \tag{3.4}$$

Why do we use $n \geq 2$? We do not want to rely on a particular markup context. So we only want to consider those terms to be concepts that occur in more than one markup context. However, if the document collection is big enough we may as well only be interested in concepts that are found in more

than two or three markup contexts. We just want to specify that anything that can only be found in one of these contexts in a document will not be considered a concept.

We indicated earlier, that a single occurrence of a term may be found in more than one markup context. It could be that two markup contexts overlap. In this sense, the last definitions do not imply that there need to be two instances of a term in a document in order to be considered a concept of type 2.

As an example for the definition of a concept consider the three words "markup", "hyperlinks" and "specified" in Sect. 3.1. If one distinguishes text in the default font, bold and italic types as the (three) different markup contexts and selects all the nouns as index terms, then the only one of the three words which would be considered a concept is "markup". The term "specified" is not an index term (because it is not a noun), and the word "hyperlinks" can only be found in one markup context (the default font). To show that a single instance of a word can be found in more than one markup context one could introduce an additional markup context, which is everything that is marked as a mathematical formula. Applied to the current section the term "markup" can then be found in the new context and in italics simultaneously.

Figure 3.2 indicates why we would consider the two terms "MSc" and "NLE" to be concepts for the document introduced in Fig. 3.1: "MSc" can be found in the title as well as one of the headings, "NLE" turns up in some anchor text as well as a heading. In fact, both terms can also be found in the normal text of the document which makes them concepts of type 3.

Applying the above definitions, we can now reduce documents to a small number of concepts. We might find a handful of concepts in a particular document or not a single one. We will need little more to construct a domain model. The only other assumption that we will make is that if we find a particular concept in a document and this document happens to contain some other concept as well, then we will want to capture this fact by saying that these two concepts must be "somehow" related. It should be reiterated that this is by no means an attempt to uncover the type of relation between two concepts. Whether the relation is useful or not will ultimately be decided by the user who applies the domain model. More specifically, in an ad hoc search system a user may either select a query refinement term proposed by the system or ignore it. We do not want to know why the user selects such a term as long as the suggestions are considered sensible by the user. Hence, our definition of *related concepts* is straightforward:

Definition 3.2. *Two concepts c_1 and c_2 are related concepts of type n for document d, written as $rel\_concepts_n(c_1, c_2, d)$, if c_1 and c_2 are concepts of type n for document d, i.e.:*

$$rel\_concepts_n(c_1, c_2, d) \Leftrightarrow (concept_n(c_1, d) \wedge concept_n(c_2, d)) \tag{3.5}$$

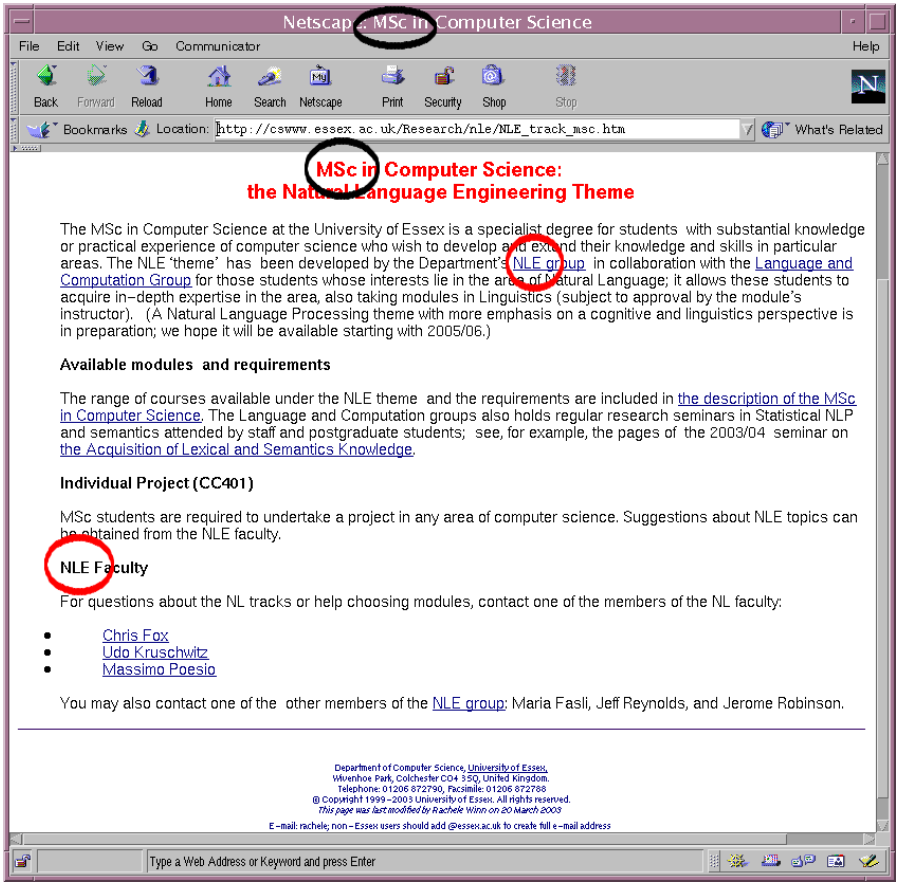

Fig. 3.2. Example concepts in a Web page

We also define:

$$rel\_concepts_n(c_1, c_2) \Leftrightarrow (\exists d\ rel\_concepts_n(c_1, c_2, d)) \tag{3.6}$$

$$rel\_concepts(c_1, c_2, d) \Leftrightarrow (rel\_concepts_n(c_1, c_2, d) \wedge n \geq 2) \tag{3.7}$$

$$rel\_concepts(c_1, c_2) \Leftrightarrow (\exists d\ rel\_concepts(c_1, c_2, d)) \tag{3.8}$$

Hence, the two concepts we identified in Fig. 3.2 are in fact related concepts because they were found in the same document.

The *related concepts* relations in (3.5) and (3.7) obviously describe equivalence relations over the domain of index terms and fixed values for type (in (3.5)) and document.

However, this is not true for the relations defined in (3.6) and (3.8). Counter examples can easily be constructed. For example, we can have one document about the *students union* and another one about the *trade union*. The term *union* may be a concept for both documents. However, each of the two documents may define a number of other concepts that can only be found in one of the documents (e.g. *bar*, *entertainment* vs. *representative*, *trade*). Hence, concepts *bar* and *union* are related concepts, so are *representative* and *union*, but *bar* and *representative* are not.

The last example gives a motivation for the next definition. We define a new relation of *vaguely related concepts* for pairs of concepts that are *not* related, but which share some other concept that is related to each of them. Why do we want to capture such a relation? It introduces an interesting aspect (discussed below) that allows us to construct a usable domain model later on.

Definition 3.3. *Two concepts* c_1 *and* c_2 *are vaguely related concepts of type* n*, written as* $vague\_rel\_concepts_n(c_1, c_2)$*, if there is a concept* c_3*, such that* $rel\_concepts_n(c_1, c_3)$ *and* $rel\_concepts_n(c_2, c_3)$ *hold, but not* $rel\_concepts_n(c_1, c_2)$*, i.e.:*

$$\begin{aligned} vague\_rel\_concepts_n(c_1, c_2) \Leftrightarrow (\ & \exists c_3\ rel\_concepts_n(c_1, c_3) \\ & \wedge rel\_concepts_n(c_2, c_3) \\ & \wedge \neg rel\_concepts_n(c_1, c_2)\ ) \end{aligned} \tag{3.9}$$

In addition, we define:

$$vague\_rel\_concepts(c_1, c_2) \Leftrightarrow (vague\_rel\_concepts_n(c_1, c_2) \wedge n \geq 2) \tag{3.10}$$

To emphasize the significance of such a relation we will consult another actual example. Processing the Essex University Web pages reveals that there is a concept *isabelle* which has among others the related concepts *proof*, *theorem_prover* and *theorem*. A second set of related concepts in the same domain is *art_history*, *gallery* and *undergraduate_studies*. Both of these sets are good illustrations of the intuitive meaning of what a related concept is. A search for either *isabelle* or *art_history* on the University Web site results in a large number of matching Web pages. But this is where the two sets of concepts differ: a query refinement step that adds the related terms for *isabelle* to the query results in exactly the same set of retrieved documents, because any document that contains the term *isabelle* contains terms such as *theorem_prover* as well. In contrast to this, documents containing *art_history* do not necessarily contain the term *undergraduate_studies*. Any of the concepts related

to *art_history* mentioned above can only be found in a subset of documents matching *art_history*. In other words, a query refinement operation that adds the concepts related to *art_history* is truly reducing the set of retrieved Web pages. The reason is, unlike the first set, the second one contains *vaguely related concepts*.

In the next sections we will discuss the domain model construction process. We will then see that sets of concepts such as *isabelle* and *theorem_prover* will end up in the same node of a concept hierarchy while concepts such as *art_history*, *gallery* and *undergraduate_studies* are the ones that are actually useful for query refinement, and they will be placed in different nodes of a concept hierarchy. Although we will not make an explicit reference to the above definition (because the model construction process does not refer explicitly to such a relation), it should become clear that we will get concept hierarchies in which the concept in the root node is related to each of the concepts in the hierarchy, but when looking at concepts selected from any two sister nodes we may well find these to be *vaguely related concepts*.

3.3 A Domain Model Based on Concepts

In the introduction we outlined the motivation for constructing a domain model. We also sketched the actual model acquisition process. In fact, we presented a slightly simplified approach. Here we will quickly revise this process and then give a formal account of the model and its construction.

Simply speaking, we constructed a domain model that was a set of concept hierarchies. Each of those hierarchies consisted of nodes represented by single concepts (see for example Fig. 3.3 which is identical to Fig. 1.6). A node was created if there were documents in the collection that matched the query represented by the particular node. Remember that what we call queries are actually just "potential" queries. We assume that the concepts that have been found in the indexing process are likely to be among those terms that users submit as real queries when they search the document collection. Based on this assumption we can simulate a user request by starting with a single concept as a possible user query (this will be the root node of one hierarchy). Utilizing the relations detected in the source data one can then explore all possible ways of constraining this query by adding a single concept to the query in a query modification step (these concepts will each end up in a daughter node of the root node). The interesting terms that could be added to the query in such a step are all the concepts that were found to be related to the original query (i.e. concept). This is an iterative process that can be applied to the new queries until eventually we reach the leaf nodes, i.e. nodes that typically represent very specific queries for which only small sets of documents can be found.

However, the original idea of creating nodes for single concepts can be improved, mainly because such a domain model would grow too large (in

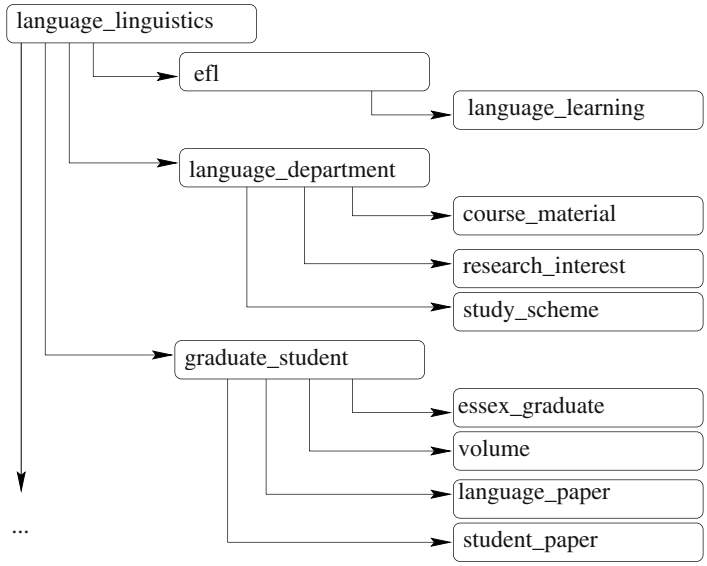

Fig. 3.3. Concept tree for the compound *language_linguistics*

particular the branching factor at each level in the hierarchy could be very high). That is not the only problem, because one could constrain the breadth of the model by reducing the number of branches that leave a node based on some ranking function. But we also construct a large number of nodes that should be merged to form a single node because they do not really represent different queries.

For an intuitive explanation of this we look again at Fig. 3.3. We assume this model has been constructed in an offline process as outlined earlier. It is now used to guide a user through a search task. The user asked for *"language and linguistics"* and the system offers a choice of three related terms that could function as query refinement options: *efl*, *language_department* and *graduate_student*. Imagine the user selects the last one in the list and is now presented with four new terms (*essex_graduate*, ...), each of which could again function as a refinement term (refining the query that now consists of the original terms and the term *graduate_student*). Why is this not good enough? The somehow peculiar relations between concepts *graduate_student*, *volume*, *language_paper* etc. were actually extracted from a collection of documents all entitled "Graduate Students Papers in Language and Linguistics, Volume ...". No matter which one of the suggested terms is added to the current query, the set of matching documents will be the same. This means the user does not really have a choice between alternative ways of getting to more specific matches. Since all those concepts frequently co-occur in the same documents we can treat them as a set of concepts rather than individually.

Hence, the definition of a domain model hierarchy that we will introduce in this chapter is slightly different from the one depicted in the introductory example, namely nodes are sets of concepts rather than single concepts. A query refinement step could now involve adding a whole set of concepts to the original query. Alternatively, the fact that a number of concepts are placed in the same node can help the users (who search or browse the document collections) obtaining an idea of what sort of documents to expect.

We will therefore refine the domain model construction process. We start by defining the necessary structures and terminology, and then move on to the actual model construction. Once we have introduced the model construction process, we will return to the example and see how the revised hierarchy looks like.

3.4 Model Structure

The basic building blocks of our domain model are concepts, nodes and arcs which are connected in a tree structure. In the definitions we refer to terms defined in the previous sections. As with the definition of concepts, we distinguish different types of domain models based on the type of the underlying concepts. That means that a domain model of type 2 contains only concepts of type 2 etc. We will start by defining the nodes that will represent the concepts in our domain model.

Definition 3.4 (Node). *A domain model node of type n, written as* $node_n(a)$*, is a set of concepts of type n, i.e.:*

$$node_n(a) \Leftrightarrow \forall c \in a\ concept_n(c) \tag{3.11}$$

We will use the terms *domain model node* and *node* synonymously. We write $node(a)$ if $node_n(a) \wedge n \geq 2$.

Conceptually, the nodes are somehow comparable to sets of synonyms (*synsets* in *WordNet*), except that the concepts forming a node do not describe a linguistic relation (i.e. *synonymy*). A node contains related concepts that typically co-occur in a document (cf. the discussion in the previous section and the earlier *isabelle* example that followed Definition 3.3). Synonyms are typically good candidates for query expansion in information retrieval applications. A user query *"taxi"* could thus be expanded to a query like *"taxi OR cab OR taxicab"* to match a larger set of documents. Analogously, we could use the concepts in a node to perform query modifications. However, we would use the terms for query refinement rather than query expansion. To refer again to the earlier example we would turn a query that contains the term *"isabelle"* into one that contains *"isabelle AND theorem_prover AND theorem AND proof"*.

Domain model nodes are the main building blocks for the structures we are actually after - concept hierarchies.

Definition 3.5 (Concept Hierarchy). *A concept hierarchy of type n is a tree consisting of domain model nodes of type n connected by weighted directed arcs.*

The direction of the arcs is such that all arcs point towards the leaves of the tree. A concept hierarchy is therefore an acyclic directed graph. We will ignore the weights for now. The reason we introduce them is to capture the fact that not all arcs are of equal importance. The weights assigned to the arcs are applied to present query modification options in a ranked order.

We will write $hier_n(h)$ to express that h is a *concept hierarchy* of type n. We write $hier(h)$ if $hier_n(h) \wedge n \geq 2$.

A hierarchy is built for every single concept identified in the indexing process; in other words, for every concept there is a hierarchy with this concept in the root node. The set of all these hierarchies will be called domain model.

Definition 3.6 (Domain Model). *A domain model M of type n is a set of concept hierarchies of type n.*

We will also define an intuitive notation of a path within a hierarchy. We will need this in order to propose strategies for applying the domain model.

Definition 3.7 (Path). *Let a_1 and a_2 be two nodes in a concept hierarchy h. There is a path from a_1 to a_2, if a_2 can be reached by traversing h starting at a_1 following the direction of the arcs.*

The notation to express that there is a path from node a_1 to node a_2 in a concept hierarchy h will be $path(a_1, a_2, h)$.

Finally, we will define the depth of a hierarchy as the maximum length of a path from the root node to any leave node in this hierarchy. The length of a path is the number of arcs traversed.

3.5 Model Construction

After introducing the necessary elements of a domain model we can now define the domain model construction process. For simplicity, we use a (very basic) Boolean approach to match documents against query terms, i.e. we assume that a document matches a query term if an indexing process has found the query term in the document. We consider every query to consist of a conjunction of query terms. We will ignore disjunctions and negations altogether.

First of all we assume a matching function which matches a set of concepts to a set of documents, in other words a function that finds matching documents

for a query. This is needed, for example, to decide whether two queries retrieve the same sets of documents.

It should be reiterated that the aim is to create nodes that contain sets of concepts rather than single ones. Each node is a set of frequently co-occurring concepts, and the idea is that such concepts do not really represent alternative query expansion terms, but adding any of them will take the user to the same set of documents. It should also be noted that the notion of *frequently co-occurring* is context-dependent. What it really means is co-occurrence in documents that contain all the other query terms as well, i.e. every concept that has been found on the path from the root node to the particular node of interest. In the earlier example we outlined that a root node *language_linguistics* would now branch to a node containing concepts *graduate_student* and *volume* among others. In some other context the concept *volume* would not be grouped with *graduate_student*; however all the documents that contain *language_linguistics* and *graduate_student* happen to contain the index term *volume* as well (and all documents that match *language_linguistics* and *volume* also match *graduate_student*).

The purpose of the matching function is to decide at each stage of the model construction process whether a new node has to be created or whether a concept can be placed in one of the already existing nodes (either a sister node or a mother node). Essentially, a new node will only be created if it represents a query that retrieves a document set which *differs* from what the other queries in this branch retrieve.

Apart from the assumed matching function we also define an auxiliary function that maps a node in a domain model to the corresponding query:

Definition 3.8 (Mapping a Node to a Query). *Let n be a node in a concept hierarchy h, and let p be the path from the root node r of h to n. We define the query that corresponds to node n as the union of the concepts in all nodes on path p.*

Remember that a node is simply a set of concepts and that we assume a Boolean model, i.e. a query is a set of conjoined query terms. The definition of a concept hierarchy guarantees that the query that *corresponds to a node* is uniquely defined. The Boolean model guarantees the same set of matches independent of the order of the query terms.

We can now express the model construction process more formally. For simplicity we will start by presenting an algorithm that leads to a model where all nodes contain only a single concept (as we have seen in Fig. 3.3). We will then specify two conditions under which new nodes are created. These conditions prevent the generation of a number of nodes, namely those that contain concepts which we want to collapse with nodes already present.

Naturally, this algorithm could be expressed in a number of ways. The constructive approach that we present here seems to be most intuitive. It follows closely the actual implementation. This is the basic construction process:

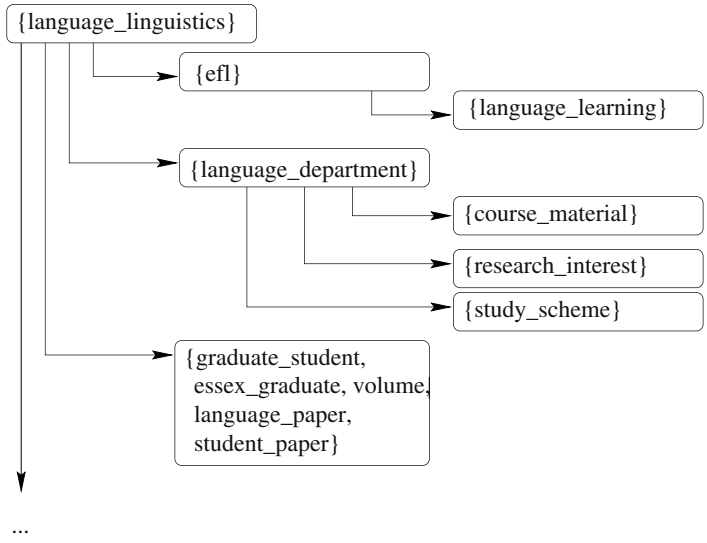

Fig. 3.4. Revised concept tree for the compound *language_linguistics*

For each concept c of type n for which a document d exists with $concept_n(c, d)$ we create a hierarchy h as follows:

1. Create a node $node_n(\{c\})$ that is the root node of h.
2. For each node $node_n(C')$ in h and each c'' with $rel\_concepts_n(c', c'')$ for every $c' \in C'$ do: Create a node $node_n(\{c''\})$ and a directed arc from $node_n(C')$ to $node_n(\{c''\})$ if the query q'' that corresponds to the new node matches a non-empty document set.
3. Perform step 2 until there are no unexplored nodes left.

Now we have to elaborate on step 2. In fact, before we create a new node $node_n(\{c''\})$ as outlined in step 2, we will first check two conditions, and if a condition holds, we perform the corresponding action:

a) If query q' that corresponds to $node_n(C')$ matches the same document set as q'', then add c'' to the existing node $node_n(C')$, and do not create $node_n(\{c''\})$.
b) If a new node $node_n(\{c''\})$ would have a sister node $node_n(C'')$, and the queries that correspond to each of those two nodes match the same set of documents, then add c'' to $node_n(C'')$, and do not create $node_n(\{c''\})$.

A revised version of the hierarchy given in Fig. 3.3 can be seen in Fig. 3.4.

There are some interesting aspects related to the model construction process that are worth noting here:

- Step 2(b) is responsible for merging two nodes that represent slightly different queries which nevertheless match the same documents (cf. the example in Fig. 3.4). Structurally step 2(b) is comparable to grouping *synonyms* as *synsets* in *WordNet*'s hierarchies.
- Step 2(a) on the other hand is not quite the same. Although the queries that correspond to the two nodes (mother and daughter node, respectively) match the same documents as is the case in step 2(b), the daughter node represents a query that contains more query terms than the mother node. But these additional query terms could be very general terms that do not constrain the original query at all. For example, adding a query term *union* to the query *"students union"* will not change the set of matching documents. Therefore we can consider the process in step 2(a) as one that identifies a relationship that is somehow comparable to *WordNet*'s *hypernym* links. What this means is that we may well add the term *union* to a node that contains *students_union* as long as we ensure that the added concept is a "very general term" that does not constrain the query (the construction process will most likely create an additional hierarchy where we find *union* further up than *students_union* which makes the comparison with hypernyms more plausible). Adding these terms may make the model easier to interpret. But if we are not interested in such a process, we will just have to add another condition in step 2 that says that we only create a new node if the corresponding query results in a smaller set of matching documents (i.e. in the definition we would require that query q'' matches a subset of what is retrieved by query q').
- Earlier on we defined *vaguely related concepts* (see Definition 3.3 in Sect. 3.2). In that context we introduced the concept *isabelle* together with its related concepts. So assuming that we run the algorithm in the Essex domain, we will construct one hierarchy that contains the concept *isabelle* in the root node. The algorithm will make sure that this root node will also contain *theorem_prover*, *proof* etc. On the other hand, the algorithm will also make sure that two vaguely related concepts will *not* be placed in the same node.
- Finally, in the practical implementations discussed later we will modify step 3 of the algorithm to perform step 2 until there are no unexplored nodes left *or* until a branch has reached a maximum depth specified in the setup.

Obviously, the constructive approach above can be described differently with the same result. What is important to point out, however, is the fact that the assumption of a Boolean model ensures that the resulting domain model is uniquely defined by the relations among the concepts and does not depend on the order in which concepts or related concepts are chosen in the construction process.

Instead of the strict notations used above such as "non-empty set", "exactly the same matches" etc. one can also imagine thresholds that would be

customizable for each domain. For example, one might want to avoid building nodes if the corresponding queries only match very few documents. Similarly one can impose thresholds in each of the other steps.

Finally, the above construction process assumes a matching function. In order to have a coherent framework that fits in with what was defined earlier, we typically define a document to be a match for a query, if all the query terms were found to be *concepts* (of the specified type) in the document. This is stricter than just assuming that the document contains each of the query terms as a keyword. This idea is in the very spirit of the book in the sense that a possibly huge amount of data can be reduced to a much simpler set of index tables.

In the model construction process we ignored the weights that are assigned to each arc. These are again application-dependent parameters. The weights express the discriminative power of a concept (or a set of concepts). These weights can be assigned as part of the outlined construction process or in a second phase of the construction of the domain model. For simplicity we can see the model described above as one in which all arcs have been assigned the same weight. In the *UKSearch* system we will assign the weights to reflect the number of matches each daughter node represents.

Practically, one can construct a domain model for each type of concepts, i.e. all concepts of type 2, 3, etc. The higher the number the more important the concepts and relations between concepts. An example application might distinguish two models, one based on type-2 concepts and the other one based on type-3 concepts. The second one is obviously smaller than the first one, since by definition any type-3 concept is also a type-2 concept. An online search system that tries to help the user by offering query refinement choices could first check the *most important* concepts, i.e. type-3 in our example. If not successful, *less important* concepts are investigated.

Such an approach represents a simple *back-off* strategy comparable to the ones involving *n-grams* in statistical language modelling [80]. For example, a language model based on trigrams can be much more accurate and reliable than one that uses bigrams, however data *sparsity* is one of the fundamental problems in statistical NLP. The *back-off* strategy in this case would work as follows: if the language model does not contain any counts for a particular trigram, then one could back off to bigrams. We do the same: type-3 concepts are considered more reliable but much more sparse than type-2 concepts. If we cannot match the query terms against type-3 concepts we back off to lower order concepts. We will discuss such a back-off approach when we introduce the model construction process for the *UKSearch* system (Chap. 6).

3.6 Using the Model for Query Modification

We want to briefly introduce some more terminology that will indicate how we apply the automatically created domain model. As a reminder, the constructed

hierarchies can be interpreted as query refinements: if a user starts with a query that matches a concept in the root node of one of the hierarchies, then we can interpret concepts in the daughter nodes as new query terms that - added to the original query - will lead to more specific matches. In fact, we could consider any concept placed in a node below the root node to be a term that could refine the original query. This will be discussed in detail when we introduce the applications. For now we will only define these relations.

Definition 3.9 (Potential Query Refinement Term (1)). *Let h be a concept hierarchy with root node root in a domain model M. A concept c_2 is a potential query refinement term for a concept c_1 iff $c_1 \in root$ and there is a node n such that $c_2 \in n$ and $path(root, n, h)$.*

Definition 3.10 (Potential Query Refinement Term (2)). *Let h be a concept hierarchy with root node root in a domain model M. A concept c_2 is a potential query refinement term for a set of concepts C_1 iff there is a node n with $c_2 \in n$ such that $path(root, n, h)$ and for every $c_1 \in C_1$ there is a node n_1 on this path such that $c_1 \in n_1$, and there is a $c \in C_1$ with $c \in root$.*

Applied to the concept hierarchy in Fig. 3.4 we would consider any of the terms sitting below the root node to be a potential query refinement term for *language_linguistics*. Hence, if a user submits the query *"language and linguistics"*, our domain model would deliver a number of possible query refinements. A ranking function would then select the most appropriate ones (a point we will get back to when we talk about the dialogue system which applies the domain model).

Similarly, we can interpret the model as a tool that can suggest ways of making the query more general. We might want to create a new query by substituting a concept in a root node for concepts found in its daughters (given that these concepts formed the original query). A user could then be suggested to replace the original query *"efl in the language department"* by *"language_linguistics"*.

Definition 3.11 (Potential Query Relaxation Term (1)). *A concept c_1 is a potential query relaxation term for a concept c_2 iff there is a concept hierarchy so that c_2 is a potential query refinement term for c_1.*

Definition 3.12 (Potential Query Relaxation Term (2)). *A concept c_1 is a potential query relaxation term for a set of concepts C_2 iff there is a concept hierarchy so that each $c_2 \in C_2$ is a potential query refinement term for c_1.*

All four definitions express formally what sort of concepts we want to consider to be refinement or relaxation terms. The dialogue manager will need to compare the user query with the hierarchies in the domain model and find out which concepts are useful suggestions for query modifications. The focus will be on *potential query refinement terms* and *potential query relaxation terms*.

3.7 Implementational Issues

Implementational details will be discussed in later chapters. However, for all the example applications that have been implemented and which access documents mainly written in English we have adopted some standard techniques for selecting potential conceptual terms in the first place. These are the only language-dependent parts. The steps are as follows:

- The documents are preprocessed into some standard format. That includes the deletion of control characters, a preprocessing step for the part-of-speech tagger etc.
- All text independent of the markup context is passed to a part-of-speech tagger that assigns a syntactic category to each word. The Brill tagger has been used for this purpose [18, 19].
- Not all words are selected as index terms. We only use those words that were tagged as nouns and furthermore select a number of noun phrases (the idea is to use patterns that are suitable to identify collocations in documents). We consider nouns and noun phrases to be the most useful phrases for retrieval tasks. For the detection of noun phrases we look for particular patterns, i.e. sequences of part-of-speech tags based on the algorithm for the detection of terminological terms described in [76]. This algorithm identifies technical terms in running text. It is effective both in terms of precision and recall regardless of the domain. The underlying observation is that most topically important terms in running text are noun phrases that follow very simple patterns consisting of nouns, adjectives and prepositions only.
  Patterns of length two and three form the vast majority of terminological terms according to [76]. There are two admissible patterns of length two and five of length three as can be seen in Table 3.1 (where A is an adjective, P is a preposition and N is a noun). The examples are drawn from the *University of Essex* sample domain and converted into a normalized form.

Table 3.1. Index terms: part-of-speech patterns

Pattern	Examples
A N	*artificial intelligence*
N N	*computer science*
A A N	*human red blood*
A N N	*electronic systems engineering*
N A N	*research active staff*
N N N	*ieee computer society*
N P N	*department of history*

For our implementations we adopted these patterns with slight variations (for example to accommodate for part-of-speech tags such as FW (*foreign word*) found in the Penn Treebank tag set [135] which we treat as either an adjective or a noun). We make no distinction between proper nouns and ordinary nouns.

It should be noted, however, that the indexing strategy outlined here has its shortcomings. If we only index on nouns and particular compounds, we will have problems with matching queries like *"Pretty"* (a member of academic staff at Essex University which the tagger is likely to tag as an adjective) or *travelling to university* (if we apply no stemming we are unlikely to find a match for *travelling* because it will be tagged as a verb). But note also that this only means that we ignore some terms or phrases that could be interesting in the domain model construction process. A user query such as *"Pretty"* can still be dealt with: The domain model is typically combined with a standard search engine, and the query refinement options derived from the domain model are presented alongside the results returned from the search engine. In this particular case there are simply no refinement options; the user will only see the search engine results.

Furthermore, the above patterns do not include compounds longer than three words such as *former Yugoslav president Slobodan Milosevic*. Here we do not consider longer patterns purely due to memory restrictions in our existing system environment.

4

Incorporating Additional Knowledge

Apart from the basic markup of the documents, other knowledge may be available. Some of this knowledge can be incorporated in the domain model construction process, and other sources may be used by the dialogue system in a similar fashion as a domain model.

What we did earlier in the domain model construction process was to impose a classification structure on a collection of partially structured documents, a structure that already exists in sources like classified directories. This is where the original idea for the domain model comes from.

4.1 Internal Knowledge

While it is very interesting to uncover implicit structure, it is as important to use existing explicit knowledge if available. That includes classifications already assigned to advertisements in a collection like the *Yellow Pages*. A lot of this knowledge is valuable as well as reliable since it has been processed in some form by humans, in the example of *Yellow Pages* classifications. Existing systems do not typically go as far as to combine explicit and implicit information, although earlier we did refer to a number of approaches that do one or the other, e.g. clustering (implicit knowledge) or classification (explicit knowledge). *Yahoo!* for example used to rely entirely on a carefully constructed classification hierarchy, i.e. explicit structure. One of our sample applications (*UKSearch*) focuses on implicit structure, and the other one on explicit structure with an option to add a domain model constructed from implicit structure (YPA).

Part of a typical classified directory consists of cross-references that relate classifications to each other. An example is a link from classification *Zoo* to *Tourist Attractions*. Clearly it is hard to create this link automatically based on linguistic information, since this is to a large degree *world knowledge*. Also a link from *Garage Services* to *MOT* is only useful in a given context. In some countries there is no such concept like an MOT certificate. In the search

process it would be advantageous to include as much of the available explicit knowledge as possible.

We will propose a fairly generic simple dialogue system that can handle other knowledge sources, we will then present two example applications that both implement the dialogue manager but differ in the type and amount of knowledge sources applied in the search process.

We will not discuss extensively how and which knowledge can be incorporated. We only introduce the relations that capture the structure between documents and classifications that are actually applied in one of the prototypes, the YPA system (see Chap. 8). All these relations are based on explicit and implicit internal knowledge.

We start by describing the classification structure and then briefly look at the relations between single documents.

We assume there is a function $classify$ from the set of documents in our collection to a set of atoms (e.g. *"Zoo"*, *"Garage Services"* etc.). We will call these atoms *classifications* (depending on the context we may also call them *categories*, *business classifications* or *headings*), and we will write $classification(c)$ to express that c is a classification (e.g. $classification(Zoo)$ and $classify(d_1) = Zoo$ for some document d_1). We can then define:

Definition 4.1. *A classification c classifies a document d explicitly, written as $classify\_explicit(d, c)$, if $classify(d) = c$, i.e.:*

$$classify\_explicit(d, c) \Leftrightarrow classify(d) = c \tag{4.1}$$

This gives us a formal definition to map documents to classifications. But usually classified directories have more structure than this. Other typical structures are links from one classification to another one (e.g. *Cameras: see Photographic equipment*). We capture this by introducing a relation $cross\_reference$ between two classifications. We will write $cross\_reference(c_1, c_2)$ to express that a cross-reference exists from classification c_1 to classification c_2. We can now define a further type of classification:

Definition 4.2. *A classification c_2 classifies a document d implicitly, written as $classify\_implicit(d, c_2)$, if there is a classification c_1 that classifies d explicitly and there is a cross-reference from c_1 to c_2, i.e.:*

$$\begin{aligned} classify\_implicit(d, c_2) \Leftrightarrow (\exists c_1 \; & classify(d) = c_1 \\ \wedge \; & cross\_reference(c_1, c_2) \;) \end{aligned} \tag{4.2}$$

Definition 4.3. *A classification c classifies a document d, if c classifies d implicitly or explicitly, i.e.*

$$classify(d, c) \Leftrightarrow (classify\_implicit(d, c) \vee classify\_explicit(d, c)) \quad (4.3)$$

The definitions so far seem to be merely another way of describing the structure that is already available in sources such as classified directories. A closer inspection reveals that they are more general. First of all, we do not have to use existing cross-references, but instead we could construct them automatically based on some similarity measure (e.g. based on term frequency, concepts etc.). Secondly, the classifications do not necessarily have to be expressed by natural language phrases but could be more abstract such as integer values (both of these issues arose when we used the YPA system with *Talking Pages* data rather than *Yellow Pages* data, the discussion of which goes beyond the scope of this book). Thirdly, and most importantly, imagine we substitute the term "classification" by the term "concept" (of some type). As a result we can construct a domain model as described earlier. We only need to consider the following three aspects:

- We turn a classification into a set of concepts by either treating the classification as one concept or breaking it down into a set of nouns and noun phrases.
- We treat cross-references between two classifications as a relation between the concepts extracted in the first step.
- This gives us the notion of "concepts" and "related concepts" to construct a domain model in exactly the same way as we have seen in the previous chapter.

To be more precise, we will start with explicit classifications. If we do have an existing classification structure (e.g. names of business classifications), then we can select nouns and noun phrases and use them as explicit classifications. We can write $classify\_explicit(d, c)$ to express that d is explicitly classified under (concept) c. For example, an advertisement d could be classified under the heading *Cameras*. We could then write $classify\_explicit(d, cameras)$, if we treat *cameras* as a concept.

Using concepts instead of classifications we could then define cross-references as follows:

$$cross\_reference(c_1, c_2) \Leftrightarrow rel\_concepts(c_1, c_2) \quad (4.4)$$

This gives us the following variation of Definition 4.2:

$$\begin{aligned} classify\_implicit(d, c_2) \Leftrightarrow (\ & \exists c_1\ classify(d) = c_1 \\ & \wedge\ rel\_concepts(c_1, c_2)\ ) \end{aligned} \quad (4.5)$$

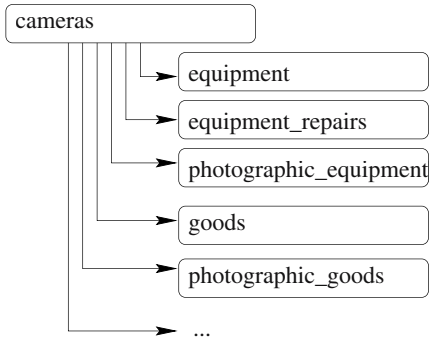

Fig. 4.1. Concept tree for classification term *cameras*

Similarly, we will then say that a concept c classifies a document d if c classifies d explicitly or implicitly as we have defined it in Definition 4.3.

The only significant difference between real classifications and our automatically created domain model is that the relation between concepts is symmetric while in general cross-references are not. However, we will treat them as symmetric, which means that a link from a classification *Cameras* to *Photographic Equipment* will also be treated as a link the other way round. As a result of this model construction process we obtain the same structures that we have seen in the last chapter (see Fig. 4.1 for part of an example concept hierarchy - nodes are respresented as single concepts).

The structure imposed by classifications and cross-references can also be utilized to introduce a cross-reference relation on the document level, but we would call this a *link* relation in accordance with common usage of that term. We could capture this type of internal knowledge with the following definitions.

Definition 4.4. *An explicit link exists from document d_1 to document d_2 if both documents are explicitly classified under the same classification c, i.e.*

$$\begin{aligned} link\_explicit(d_1, d_2) \Leftrightarrow (\ & classify\_explicit(d_1, c) \\ & \wedge\ classify\_explicit(d_2, c)\ ) \end{aligned} \tag{4.6}$$

Definition 4.5. *An implicit link exists from document d_1 to document d_2 if d_1 is classified under c_1 and d_2 is classified under c_2 and there is a cross-reference from c_1 to c_2.*

Definition 4.6. *A link exists from document d_1 to document d_2 if there is an implicit or an explicit link from d_1 to d_2.*

These definitions simply propagate the classification structure down to the document level by introducing links between individual documents classified under "related" business classifications.

Finally, we could combine this with explicitly expressed links within documents (e.g. hyperlinks in Web pages or citations in research papers).

We will however not explore this issue, because document level relations tie the domain model closely to the document collection. We, on the other hand, focus on a model that can be used independently of the individual documents. Therefore, we will not elaborate on such relations.

4.2 External Knowledge

We may also want to incorporate some external domain knowledge in the outlined approach. For example, *WordNet* might not be suitable as the only source of linguistic knowledge in a search task, but it proves to be useful as one of a number of models.

The structure of *WordNet* makes it easy to handle it like an automatically generated domain model. We can actually make use of such external knowledge. For the practical applications presented in the next part of this book we explored *WordNet*'s synonym, hypernym and hyponym relations. The links describing these types of semantic relationships are comparable with the tree structure in the domain models we discussed so far (although *WordNet* relations are explicit linguistic relations). Figure 4.2 is an example of some of the knowledge encoded in *WordNet* for the noun entry *camera*. Displayed are the hypernym relationships for that entry.

We incorporate external knowledge sources by means of a ranking function that judges the relevance of different types of relations in a given context. This is part of the customization step necessary to set up the dialogue manager.

Apart from some readily available external knowledge or other suitable sources as characterized above one can also construct domain models using different techniques (i.e. not based on document markup). For example, extracting conceptual information or relations between terms based on linguistic or word frequency information may result in additional domain models. At this stage we see this option as a potential back-off technique, i.e. to be used if the markup-based extraction fails to deliver useful results. It remains part of the future research to see which technique works best in a given situation and how to best combine a number of extraction techniques.

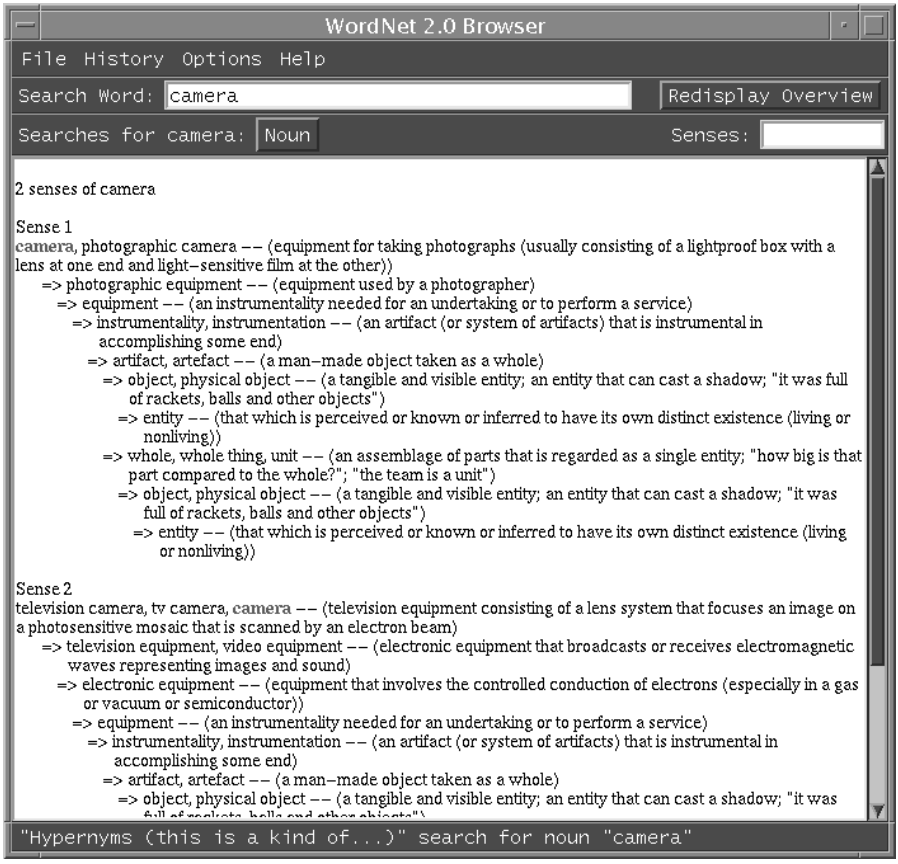

Fig. 4.2. *WordNet*: some knowledge encoded for *camera*

5

A Dialogue System for Partially Structured Data

Search engines are an excellent tool for locating relevant information in a matter of seconds. The user merely has to type in the request and the result is a ranked list of documents. Not even the document format seems to matter anymore. Not long ago only some specialized search engines like CiteSeer[1] [96] would have handled formats like Postscript or PDF; now it seems common practice to retrieve PDF files alongside pure text documents.

However, with an average query length of between two and three words it seems difficult to select the most relevant documents from what was estimated back in 1999 as a pool of already 800 million pages on the publicly indexable web [97]. How can a two-word query describe precisely what the user is after? This is true not just for searches on the entire Web but also in smaller domains such as the ones discussed in this book. Our solution is to initiate a dialogue exploiting domain knowledge that helps the user navigate to the set of most relevant documents. Obviously, this dialogue needs to be short and should only be started if necessary. Furthermore, the choice of how to get to the best matching documents must be left to the user. The dialogue must not dictate the next steps.

As we have seen earlier, a user searching the Essex University Web site for *"union"* might not be aware of the fact that the query is highly ambiguous. We pointed out that there are Web pages in that domain presenting information about the *trade union*, the *students union*, the *European union* and a *Christian union*. Not just that, but there is a number of pages devoted to *discriminated unions*.

We argue that a simple dialogue system should be used which is based around some domain knowledge. This chapter will lay the foundations of the dialogue system that accesses the domain models described in Chap. 3 and incorporates other available structures as indicated in Chap. 4.

It should be stressed again that what we mean by "dialogue" in this context is a fairly narrow interpretation of the concept. We are interested in

[1] `http://citeseer.ist.psu.edu`

information seeking dialogues, interactions between users and a system where the system is to provide the information the user is interested in. The dialogue steps (or dialogue moves) are transitions between fairly simple dialogue states. Our dialogue system is based on a few fundamental ideas, some of which have already been introduced, the others will be discussed in this chapter:

- The document set defines the concepts available. These concepts are structured in a domain model. This is done in an automatic offline process.
- The concepts (and any other user query terms) are used to express the user query formally. We will call such a formalized query a *goal description*.
- The dialogue can be seen as a movement in the space of goal descriptions. The system always tries to present suggestions of how to change the current user request. These suggestions are presented as possible options for a user to choose from and they typically represent refinements or relaxations to the original user query.
- Goal descriptions are translated into queries to the search system. The queries represent a list of results. The relation between a goal description and queries defines refinement and relaxation steps that drive the dialogue manager.

5.1 Dialogue as Movement in Space

A dialogue between a user and a system can be seen as a sequence of steps taking the user from an initial state to a final state in a script that predefines permissible steps. However, in this context dialogue is best defined as a movement in the space of descriptions of (the content of) documents. Those descriptions act as constraints that can be specified or relaxed in the interaction between the system and the user. The ultimate goal is to find an appropriate set of documents in a collection (represented by a set of descriptions) that matches the user's information need.

Descriptions of documents can include properties of single documents as well as relations between documents, which can all be abstracted as a set of basic categories like word, classification and linkage information. How these properties show up or how they are encoded depends largely on the particular domain. For example, containing a certain keyword can be a property of single documents. Other such properties can be keywords found in the title, or information about document type or classification etc.

Relations between documents could be represented by explicit or implicit links between two documents (e.g. hyperlinks in HTML documents) or by the fact that two documents are stored in the same directory.

We want such a dialogue between a system and a user - which exploits descriptions of documents - to satisfy at least the following criteria (this is the same list of informal requirements that we had identified in the introductory chapter):

- allow the user to refine or relax the information request if necessary
- take as little as possible (in terms of dialogue steps)
- present the user a number of choices to continue the dialogue if this is appropriate
- present the best possible choices only
- avoid unnecessary dialogue steps.

When will a dialogue step be necessary? If the user's information request can be satisfied by a set of matching documents, then these matches can be presented and from the system's perspective no further dialogue steps are necessary. However, the user may not be happy with the answers or might want to explore some of the query modification options that have been suggested by the system. Therefore, the user should always be allowed to continue the dialogue.

Whether the system always displays the best matching documents alongside query modification options or whether such matches will only be displayed once the information request is specific enough depends on the system setup. If we search a Web site, then we may want the system to behave like a standard search engine. The best matching documents would then be presented alongside query modification options suggested by the system. Search in a classified directory however will probably have a more sophisticated dialogue component, and no documents (i.e. advertisements) are displayed unless the information collected from the user has been specified to a certain degree.

To be more concrete, we want a dialogue system that will satisfy the criteria outlined above by navigating a user through the available documents. This dialogue is driven by the domain model.

5.2 Dialogue Example

A simple example to illustrate our dialogue system is the system's response to a user query that retrieves a large number of documents. We will concentrate on deciding which query refinement terms are useful to offer the user in this case. The only document description our system deals with is one that describes whether a keyword is or is not an index term for a document.

We assume the query to be a set of terms. Let the current query describe the set of documents called *Cluster 1* which is a subset of our document collection (depicted by Fig. 5.1). The next step is to determine discriminating terms that retrieve subsets of *Cluster 1*. In the simplest case we only consider concepts. All concepts related to any of the current query terms are considered as candidates. We can consult the domain model and assess the trees whose root nodes contain query terms. As a result the dialogue step would involve the display of the "best" concepts and the user can choose one or can select some other possible option. Because we are only dealing with concepts and not keywords in general, this is a feasible task that can partially be performed offline as described earlier.

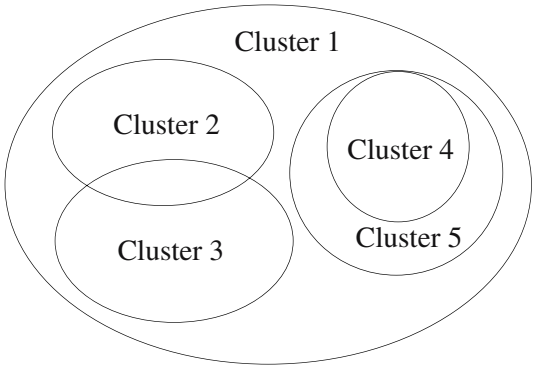

Fig. 5.1. Clustering the potential results

What are the "best" concepts? The example in Fig. 5.1 represents exactly four potential concept terms applicable to the current query. Each of the sets refered to as *Cluster 2...5* are the sets of document defined by adding exactly one of the potential concept terms to the current query. Now we consider all those concepts which result in non-empty document sets. It is obvious that the created document clusters can overlap. In the introductory example the terms *students_union* and *christian_union* are such a case. Furthermore, the partitioning of the document set is normally not exhaustive, because the number of concepts is fairly restricted (simply because we do not want to present more than a certain number of options to the user). The dialogue should help the user find the actual documents he or she is interested in. It seems intuitive to present *some* possible ways to refine the query, preferably picking good discriminators. From what was said so far, the concept terms representing each of the clusters in the example would be possible user choices. However, we do not want *Cluster 4* and *Cluster 5* being presented side by side. Instead, the larger document set is selected as a choice alongside the option to refine the query even further. All this is exactly what the domain model gives us!

Now the user has to decide how to continue the dialogue. The discussed options will be displayed alongside some standard choices which include:

- Use the text input field to modify the current query.
- Display the best matches only (if this has not already been done so as part of the initial query processing step).

The text field is necessary to give the user the freedom to decide what to do next. If the terms that are offered by the dialogue system are not relevant for the current search request, then the user might want to provide additional information. The user could also use the input field to ask for help, restart the dialogue or relax the current query (as for example in *"No, I want the students union Web page."*).

The user can also select one of the default choices as listed above. This is not an exhaustive list. The customization of the dialogue system can introduce a number of very detailed options. For example, the dialogue component in a directory enquiry system such as the YPA (see Chap. 8) might offer the user to display all results that could be found by searching for company names if this is appropriate, or it could list a number of relevant directory classifications.

Note that user and administrator have some freedom of defining the maximum number of possible choices to be presented. The dialogue system will rank all choices and display only those with the highest relevance values. Once a user has made a choice, the process of calculating possible query modification suggestions starts again and a dialogue step is performed.

5.3 Static *vs.* Dynamic Clusters

The task of the dialogue manager is to guide a user to a cluster of relevant documents as a result of a search task. Potentially interesting clusters of documents representing relaxation or refinement steps are being investigated and evaluated by the system each time the user submits a query or makes a choice. These clusters are defined by document descriptions, i.e. a number of variables to describe different parameters of the documents to be retrieved.

However, as we have stressed in the introduction, there are really two types of document clusters that we distinguish: the ones that can be created *offline* and the ones that have to be constructed *on the fly*. Earlier, when we discussed the domain model construction we actually explained how the *offline* clustering works. Concept terms are used to modify queries consisting of other concept terms and the results of that are encoded in the domain model. As long as the user submits queries that consist of terms identified as concepts in the indexing process, a simple lookup in the domain model will retrieve a number of query modifications without the need of any online processing. After all, the domain model is based on the idea of organizing discriminating terms so that the domain model is a useful tool to be applied in the search task.

But when dealing with real user queries we will not be able to rely entirely on what is encoded in the domain model. We have no idea what queries are going to be asked and how the user decides to continue in each interaction with the system. Therefore we will have to consider creating other options *on the fly* by consulting the available knowledge sources.

5.4 Real User Queries

To illustrate the dialogue on realistic examples we pick some of the queries most frequently submitted in our Essex University sample domain (using a version of the implemented *UKSearch* system prototype which differs from

the version explained later in that it accesses its own database and not a search engine). When looking at the examples it must be kept in mind that no manual modification was applied to the resulting concept structure (the database used is a version that is older than the one reported elsewhere in this book, and all the options we will see are query refinement options).

Let us first go back to the introductory example. The user asked for *"union"*. There is a large number of matching documents. Let us assume the threshold of the maximum number of matches set by the administrator in this case is 50, i.e. if more than 50 matching documents are found, the system will initiate a dialogue (i.e. "too many" matches). Alongside the default choices the search system offers four concepts, each of which could constrain the original query: *trade_union*, *student_union*, *christian_union*, *discriminated_union*. The user selects *students_union* (another frequent query submitted to the search engine at Essex University). Say there are still "too many" matching documents. The choices generated by the system are as follows:

- Display the most significant matches only.
- Constrain the query by adding one of the following terms:
  - *bar*
  - *welfare*
  - *entertainment*
  - *shop*
- Supply more information using the input field to constrain the query.

At this stage the user decides to see the most significant matches only. The 37 best matches are displayed (in this particular case we retrieve all those documents for which the query terms were found to be concepts).

Let us look at two more examples, both from the top ten most frequently submitted requests according to the Web server's log file (recording submissions over a period of several months). The most frequent query was (possibly surprisingly) one unrelated to the university: *"yahoo"* (it needs to be pointed out that the log files were recorded in 2001, nevertheless we would have expected some domain specific query to come top of the list).

Figure 5.2 displays the concepts that the dialogue system would propose based on the domain model if the original query was *"yahoo"*. That means each of the four related concepts is a possible choice to constrain the query leading to more specific results. Of course, the user is free to add any other query terms in the input field provided.

Another example from the top ten most frequent queries is the user request for a *"prospectus"* as displayed in Fig. 5.3. This is another example that shows the significant differences between a domain-independent world model and the one extracted automatically.

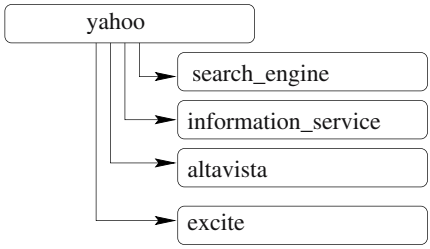

Fig. 5.2. Query refinement options for the example query *"yahoo"*

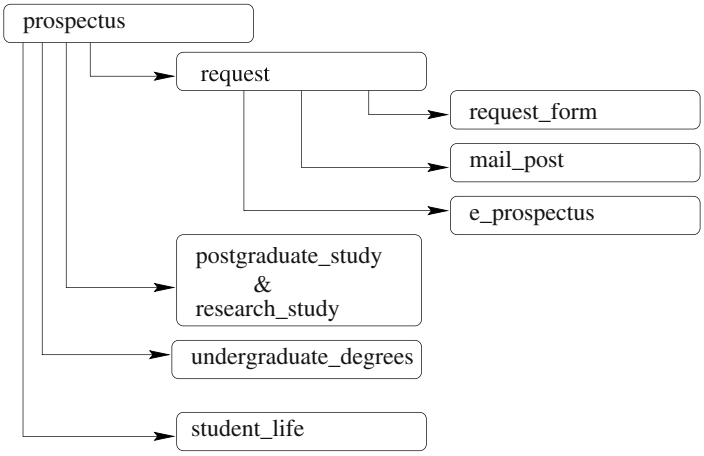

Fig. 5.3. Query refinement options for the example query *"prospectus"*

5.5 Properties

The dialogue between user and system is formally best described in a bottom-up fashion. We will first define the essential elements needed to formalize a dialogue state starting with document and system properties. These properties are used to capture more formally what the user is after. They basically encode the actual user query, but can include other information such as certain parameters of the search engine that the user can specify. All these properties with their appropriate values give us a formal representation of the user request, which we will refer to as a *goal description*.

Based on these definitions we will characterize dialogue steps, dialogue strategies and the scope for customization (something discussed in more detail in the later chapters on practical applications).

5.5.1 Document Properties

First of all, we define properties of documents which are used to match user queries against the documents in the collection.

A *property of a document* is an attribute of a document that can be assigned elements from a suitable domain as its argument.

An example property of a document can be: "the document contains a set of keywords". Let the domain be the power set of the set of all keywords in a document collection. In that case a possible value in this domain might be $\{essex, university\}$. One needs to define a "matching function" for this particular property that decides for any given document in the collection how well the attribute-value pair (e.g. $keyword(\{essex, university\})$) matches the document. We could simply assign truth values, but we want to keep things more general to assign for example a ranking value that characterizes how well a particular value of a property matches a document. In the following we will consider the interval $[0, 1]$ of real numbers as codomain for any of those functions (and treat *true* as 1, *false* as 0).

Note that this is no attempt to introduce *fuzziness* just for the sake of it. It is rather a pragmatic move based on the actual applications we have in mind. By allowing values that are not just *true* or *false* we can incorporate standard information retrieval measures such as term frequency and inverted document frequency or other information we can derive from the data sources. In the above example we will be able to distinguish a document that merely mentions the phrase "University of Essex" from something like the home page of the Essex University Web site.

We define a *document description* as a set of attribute-value pairs where each attribute is a property of a document and its value is an element from the domain of that property.

5.5.2 System Properties

The terminology we have introduced so far should be sufficient to formally express the user's information needs in respect to the documents. In addition to that, we introduce another set of properties that specify *how* the user needs are to be matched against the database in the search system. When we refer to the *search system* (or *system* for short) we mean the apparatus (implemented or not) that has been set up to search a particular document collection; e.g. the YPA and *UKSearch* are two such search systems.

A *property of a search system* is a parameter of the system that can be assigned elements from a suitable domain as its argument.

We use system properties to encode parameters that control the search process, for example parameters that define the search space, the domain models etc.

System properties do not just work as simple parameters, they include references to the world models to be applied and a number of system specific

settings. Some of those properties are "read-only", i.e. a user cannot change them, others will be changed automatically or can be set by the user explicitly. For example, in the default YPA system the user cannot modify the type of world models that are consulted in the search process. On the other hand, the system might produce an option that the user can select which says *"Show me addresses that were found by searching company names as well."* If the user selects this option, then the request will be handled with the system properties adjusted appropriately.

A *system description* is a set of attribute-value pairs where each attribute is a property of a search system and its value is an element from the domain of that property.

5.5.3 Goal Description

A *goal description* is a set of attribute-value pairs where each attribute is a document property or a system property and its value is an element from the domain of that property.

This last definition is the framework to comprehensively describe a user's information need in respect to the required documents as well as the desired system parameters. A goal description is something like a formalized user query. At each stage of a dialogue we can have a look at the goal description and see the current user request. In the simplest case we would just have a list of keywords. In a more advanced search system like the YPA we have a number of properties that can be tuned and changed during the dialogue, including the system properties to represent the search strategy:

- Should all index tables (including the business names) be searched? Or just the free text?
- Should an external world model be applied? If so, which relations?
- If a world model is to be used, should it always be applied or only if the search without applying the world model does not find "enough" documents?

Possible world models could be linguistic knowledge sources such as *WordNet*, domain specific ontologies, the domain model derived from the existing classification structure, or some commonsense knowledge. We do not want to specify that here. In the YPA we allow the user to switch the use of a world model on or off. If it is switched on, the system accesses the automatically constructed domain model (as described in Chap. 4) as well as *WordNet*'s synonyms to suggest query modifications.

Some document properties in the YPA are (note that the documents in the YPA system are classified advertisements):

- Keyword x can be found in the business name of the advertisement.
- Keyword x can be found in the business classification.

We have not defined how a goal description is matched against documents in order to retrieve a set of documents. The reason is that the actual query format depends on the search engine, also the expressiveness of a query will depend on the capabilities of the search engine. A simple search engine will only allow a list of keywords as a query. A more advanced search engine will encode other properties (as in the example of the YPA).

5.6 Dialogue

The interaction between user and search system serves the purpose of taking the user from some initial search request to a satisfactory set of answers. In other words, the *goal description* is constantly being updated in a sequence of simple *dialogue steps* until it matches a set of documents the user is happy with (or until it has been established that no suitable matches exist for the user's request). A dialogue step can be seen as a number of smaller steps to be performed in this order:

- evaluate the user input
- calculate the new dialogue state
- perform all actions corresponding to the state transition (e.g. display results and choices).

It was said earlier that we understand dialogue roughly as a movement in the space of document descriptions. That means there are - in the most rigid sense - no strictly defined dialogue states as they are commonly used in dialogue systems. Our dialogue states are calculated automatically, and as we will see they are represented as a tuple containing a number of different types of information.

However, having said that, at a high level we can map our dialogue system to a state-based system that is extremely simple. In the following we will first look at this simple abstraction and then move on to describe dialogue states at a more detailed level.

5.6.1 High Level Dialogue States

The dialogue is handled by a dialogue manager. To break down the tasks of a dialogue manager we can consider it to consist of two domain-independent parts that can be customized to fit a particular application: a *core dialogue manager* and a *default dialogue manager*.

The core dialogue manager covers all tasks to be handled by any similar dialogue system without having to access the *database system*. This would detect:

- that a user wants to *quit* (or *restart*) the dialogue
- *meta queries* (where the user asks for some help etc).

The *default dialogue manager* is this core engine expanded by adding coverage of the other states that can occur in information seeking dialogue systems, namely:

- a database access is *successful*
- a database access results in *too many matches*
- a database access results in *too few matches*
- *information* is *missing* (a submission to the database requires more information)
- the user provided *unknown input*, i.e. the dialogue manager detected some input that cannot be interpreted (this state typically triggers a clarification step)
- some *inconsistency* occurred, i.e. the dialogue manager encountered an error (e.g. the database system is not available).

This outline compares to the two-layered dialogue architecture in [2], where the *default dialogue manager* covers the upper layer of dialogue states, and where customization may refine those and add a second (domain-dependent) layer. Differences, however, are the set of dialogue states and the distinction made between the various possible states.

The set of dialogue states above is fairly small and proved sufficient for the YPA system. But we can reduce this set of states to an even smaller set if we assume that in our system the best matches for the current goal description are *always* displayed, whether there are thousands or no matching documents. This compares to what standard Web search engines do, except that we enrich the result by presenting potentially useful choices that a user can select to continue the search along a particular direction.

The way we reduce the set of outlined dialogue states to an even smaller set is by treating a number of different states as special cases of some more general state that we call the *Display state*. Conceptually, we do not want to distinguish between a database access resulting in too many or one resulting in too few matches. In each case the new dialogue state needs to encode a set of matching documents and a set of possible options; only in one case the options will mainly consist of potential query refinements and in the other one it will be query relaxations. Therefore, the two states expressing *too many matches* and *too few matches* are just instances of the *Display* state. Similarly, a state representing a successful query is conceptually no different: the state is characterized by some set of matching documents and potential options to continue. This may sound as if we are conflating states here that should really be treated differently. In fact, in the YPA system we do treat the three states differently. But look at a dialogue-based Web search engine in which we want to clearly separate the dialogue manager from the index database, because we may not be interested in building our own search engine and instead just incorporate an existing one that makes sure the database is always kept up-to-date. The dialogue manager has access to the automatically acquired domain model and should not be linked too closely to the search engine to find out

whether there are any matches for a particular query or not. The dialogue manager will therefore not "know" how many matches can be found for a user query. Yesterday the search engine may have returned 2 documents, today it could be 50. The dialogue manager should calculate suggestions (derived from the domain model) that would be sensible for both cases, i.e. options for making the query more specific or more general. The user will decide which ones are most sensible. *UKSearch* is an example of such a dialogue-based search system for Web documents as we shall see later on.

We can therefore abstract all the necessary dialogue states to be expressed as an instance of one of the following states:

- *Start*: The dialogue starts in the *Start* state.
- *Display*: The *Display* state can be considered the default state in that any user input typically returns a result. This result will be the best matching documents together with choices that the user can select from to continue the dialogue.
- *Meta*: The *Meta* state represents a user request for some meta information.
- *Unknown Input*: The system does not know how to interpret the input. This might require some clarification dialogue.
- *Missing*: More information is required to access the database. In some applications it is necessary to provide a minimum of information to move into the *Display* state, e.g. to find a classified advertisement it is not sufficient to specify the location or payment method only.
- *Inconsistency*: Some inconsistency occurred.

A transition is possible from each state to any other state. We do not claim that this set of states is in some sense universal, it just seems adequate for ad hoc search tasks. In fact, Fig. 5.4 depicts the dialogue in a simplified way where we distinguish only three of the states. User input is abstracted as *Input* in the diagram. As we will see later, the *UKSearch* system only distinguishes these three states in its basic implementation.

From the system's perspective only the *Display* and *Inconsistency* states are final states. However, a user can consider any of the states a final state.

Now, the most interesting aspect in Fig. 5.4 is the transition between *Display* state and *Display* state.

In the following we will concentrate on looking inside the *Display* state and break this state down into a number of possible "low level" states that are defined by some parameterized representation of user request, dialogue history and other pieces of information.

5.6.2 Low Level Dialogue States

We will leave aside all states other than the *Display* state for now. In the practical applications we will come back to the other states, but they are of no theoretical interest in this context.

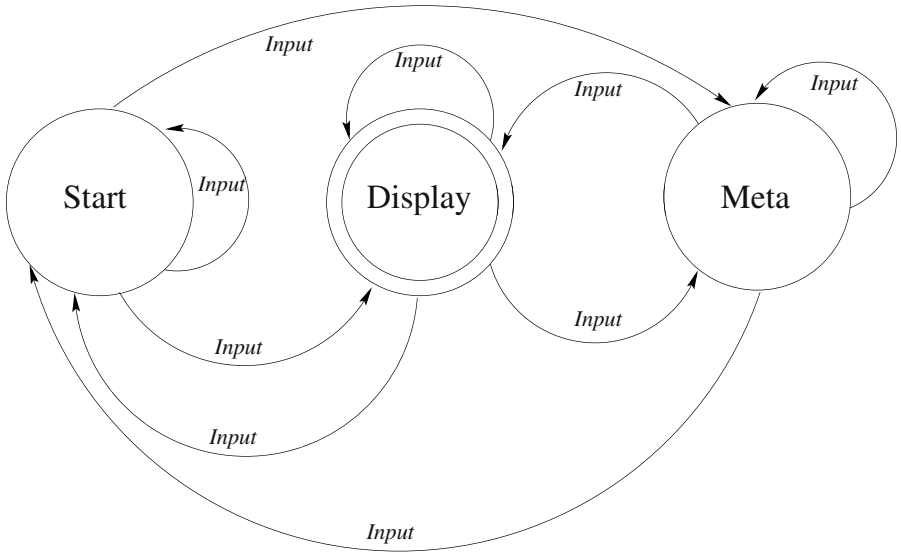

Fig. 5.4. High level abstraction of the dialogue

All the dialogue steps we investigate now are transitions from *Display* state to *Display* state. Therefore, it makes sense to look at a lower level of dialogue states, a level which distinguishes dialogue states which are all instances of the high level *Display* state.

The dialogues we describe are system initiated. Although the user has some freedom to navigate through the dialogue, it is basically an information seeking task that needs to be performed and the system is merely an assistant to help the user get to the right set of answers. As such the focus is a system that presents results alongside possible choices the user might want to consider to continue the search task.

How do we move from one state to the next one? The general idea is to analyze the current dialogue state and the user input in order to generate a new dialogue state (in other words: to perform a move). We characterize a dialogue state by:

- a dialogue history
- a goal description (i.e. the current formalized user query)
- the list of results that matches this goal description
- a list of potential user choices (options to relax, refine or change the query in other ways).

The list of choices is selected from a set of possible modifications to the goal description taking into account their effect on the result set. Again this might sound very general. The reason here is that we will be able to describe

a variety of dialogue strategies depending on how those possible choices are to be selected by the system.

Relaxation and refinement steps are the essential building blocks that define the choices a user can select from to continue the dialogue. Ideally, our dialogue system would explore all possible ways for relaxation and refinement and select the best ones. However, for two reasons we do not follow that route. First of all, the complexity involved would slow down the response time significantly, and that would make this work irrelevant for practical applications. The second reason is our experience that users are generally not very happy if the system performs actions that cannot be explained, e.g. automatic query expansion using hypernyms *and* synonyms [93]. Therefore, each of the choices represents a *single* refinement or relaxation step. As far as possible, refinement and relaxation choices are derived from the domain models that are built automatically based on the documents' markup structure.

The system should not be restricted to offering *either* constraining *or* relaxation options only. A data driven approach could well present both types of suggestions alongside each other.

A ranking function needs to evaluate each of these choices and return a ranking value that will make sure that only the most relevant choices are presented. Thresholds and cutoff points can be defined depending on the particular application.

Each of those choices can be broken down into a goal description and some input. The input is what the user is offered and selects (for example a new query term that is added to the original query), the goal description is a formal representation of the new search request that combines the new input with the goal description in the previous dialogue state. For example, assume a simple goal description format which only represents a set of keywords, and the user asked for *"union"*. A possible choice could then be represented by some phrase such as *"Do you want to add european_union to your query?"* (this is the input, i.e. what the user is offered), and a new goal description that represents a query consisting of the two query terms *"union"* and *"european_union"*.

So far we have concentrated on the list of choices that the dialogue system calculates and presents. However, the user could ignore all this and instead submit some text in the input field. An example would be if the user who asked for *"union"* cannot find any sensible suggestions and types in *"statistics"* (knowing that there is a number of documents about trade union statistics in the collection). Alternatively, the user could have input some help request. Given such an input, the calculation of a goal description and a new list of results is straightforward (but again application-dependent). The input is parsed in some appropriate way and where necessary properties of the goal description are updated using the value of these properties submitted via the user input. A list of results is constructed by turning the goal description into a query and submitting the query to the database system (to be discussed later).

We can summarize all this more formally as follows:

Definition 5.1 (Potential Choice). *Given a search system $\mathcal{S}$, we define a potential choice as a tuple $< Goal, Hist, Input >$ where Goal is a goal description, Hist is a dialogue history and Input is a user input.*

Definition 5.2 (Dialogue State). *Given a search system $\mathcal{S}$, we define a dialogue state as a tuple $< Goal, Hist, Choices, Result >$ where Goal is a goal description, Hist is a dialogue history, Choices is a list of potential choices, and Result is a list of documents.*

The definitions need some further explanation. Definition 5.2 expresses the general idea of a dialogue state that is being defined by a goal description representing the current information request, a dialogue history and the necessary return values calculated by the system *in response to* the last user input.

Note that each element of *Choices* will again contain some goal description and history elements. They will all differ from each other because each choice represents a different user input; and goal description and history represent the state after a user has selected the choice.

For the sake of abstraction we do not actually capture what *exactly* is meant by "dialogue history", "user input", "list of choices" and "list of retrieved results". The reason is that the internal representation of those elements differs from application to application. What is important however, is what elements a dialogue state (and a potential choice) consists of.

Our definition of the initial dialogue state looks as follows:

Definition 5.3 (Initial Dialogue State). *Given a search system $\mathcal{S}$, the initial dialogue state is a dialogue state $< Goal, Hist, Choices, Result >$ where Goal is set to its default value; Hist, Choices and Result are empty.*

Since we have been very nonspecific about the components of a dialogue state we have to do the same when we define the *initial dialogue state*. Being "empty" will not be specified any further since it will depend on the actual implementation what empty means. We only assume that the set of instances for each of the variables that define a dialogue state must contain some null element (i.e. the empty value). In addition to that, we also assume that a (system dependent) *default value* of a goal description exists. That could for example be the formal representation of an empty query.

The *dialogue history* can be quite a complex structure encoding all the dialogue steps that have happened so far including all the user input steps, starting from the initial dialogue state.

The *user input* can take a number of forms as well. It could be a typed input, or the output of a speech recognition component. It can also be a selection the user has made from a list of choices presented by the system (using the mouse for example).

As outlined earlier, the list of choices a user is presented with (the *Choices* argument in a dialogue state) is generated by consulting the domain models

and accessing the search engine. This is usually a multi-stage process which can include a number of calls to the search engine.

For completeness, we define two functions, one that takes a goal description and turns it into a query, and one that matches a query against the set of documents in the collection. Obviously, the technical details are not important here. It depends entirely on the search system how these functions actually do the mapping.

Definition 5.4 (Current Query Function). *Given a search system $\mathcal{S}$, we define the function currentquery as a function on the set of goal descriptions to a set of queries to $\mathcal{S}$.*

This function is a simple translation of a formalized user query into some other representation language. For example, a goal description that represents a user request for *"union"* in the Essex domain could be translated into a query to Google which might then look as follows:

```
www.google.com/search?hl=en&ie=ISO-8859-1
                &q=union+site%3Aessex.ac.uk
```

Definition 5.5 (Retrieve Function). *Given a search system $\mathcal{S}$, we define the function retrieve as a function on the set of queries to $\mathcal{S}$ to the power set of documents in the collection.*

To be more precise, we retrieve a *ranked list* of documents. What needs to be mentioned is that probabilistic retrieval systems treat every single document as a match with a possibly very small relevance value. In that case one can define some cutoff point and treat all those documents as matches whose relevance values are above that point.

Now we shall define *how* a dialogue state is calculated. Again, the customizations for particular collections will be left for later when we get to describe the example applications in detail. The underlying principles will be discussed here. We can see dialogue as a state machine in which however the dialogue states are dynamically constructed.

Definition 5.6 (Dialogue Function). *Given a search system $\mathcal{S}$, a dialogue function trans is a function that maps all ordered pairs of dialogue state and user input to dialogue states.*

We write $S_2 = trans(S_1, Input)$ to express that dialogue state S_2 is the result of applying dialogue function *trans* to a dialogue state S_1 and a user input $Input$. The process of applying the dialogue function is called a *dialogue step*.

So far we assumed that a user is presented a list of options that he or she can choose from to continue the search process if the results returned so far have not satisfied the information needs. These choices include the addition of query terms for query refinement or the relaxation of the query in some

specified way or the choice to just add some more specific information. The actual user selection is evaluated against the current dialogue state to move to a new state. However, what remains to be discussed is a methodology of how the system comes up with these choices in the first place. What needs to be done inside the dialogue system to transform the current dialogue state and the user input into a new state that contains a new set of options a user can choose from? We will show that a new dialogue state is not an element of some predefined set of permissible states but is being generated dynamically (not in the abstract but in the computational sense).

5.6.3 Constructing Potential Choices

Generally speaking, we understand refinement and relaxation as follows:

Definition 5.7 (Query Refinement). *Given a search system $\mathcal{S}$ and goal descriptions $Goal_1$ and $Goal_2$, then we define $Goal_2$ to be a query refinement for $Goal_1$ if*

$retrieve(currentquery(Goal_2)) \subset retrieve(currentquery(Goal_1))$.

Definition 5.8 (Query Relaxation). *Given a search system $\mathcal{S}$ and goal descriptions $Goal_1$ and $Goal_2$, then we define $Goal_2$ to be a query relaxation for $Goal_1$ if*

$retrieve(currentquery(Goal_2)) \supset retrieve(currentquery(Goal_1))$.

Analogously, in terms of dialogue steps:

Definition 5.9 (Refinement Step). *A dialogue step from a dialogue state $< G_1, H_1, C_1, Result_1 >$ to a new state $< G_2, H_2, C_2, Result_2 >$ is a refinement step if $Result_1 \supset Result_2$.*

Definition 5.10 (Relaxation Step). *A dialogue step from a dialogue state $< G_1, H_1, C_1, Result_1 >$ to a new state $< G_2, H_2, C_2, Result_2 >$ is a relaxation step if $Result_1 \subset Result_2$.*

The above definitions are very general, and it is questionable whether they are very useful when, speaking in terms of queries, one considers two queries that seem to have nothing in common except that coincidentally one of them matches a set of documents and the other one a subset of that. This is only a theoretical problem, because when we apply the definitions we actually look at pairs of closely related queries, e.g. pairs of queries (or goal descriptions to be precise) where one is the result of a simple modification of the other one.

For the calculation of potential choices we use the automatically constructed domain models. Instead of querying the search engine with a large number of potential query modifications it will often be sufficient to consult the domain models. As described in earlier chapters, those models are constructed in a purely data driven process. The resulting concept hierarchies

encode information about how conceptual terms detected in the document collection are related to each other, about the distribution and discriminating power of concepts in the document collection etc. Note that we say it is *often* sufficient to consult the domain models. We have to point out that it may well be appropriate to construct potential choices on the fly. Our domain models do not encode all possible relationships. We restrict the size of the concept hierarchies. Earlier we presented an example where a user originally asked for *"union"* and could not find any sensible refinement options. This user then enters *"statistics"* in the input field (knowing that there are documents about trade union statistics). Now those two terms have been identified as related concepts, but this relation never found its way into the domain model, because each of these two terms has a large number of related concepts. Most of these relations seem to be more prominent than the relation between the two concepts *union* and *statistics*.

Each query refinement or relaxation is called a *query modification* and we have to rank them in order to construct potential choices that can be presented to the user. Only the most highly ranked query modifications are chosen to turn them into potential choices. We define a function that ranks query modifications:

Definition 5.11 (Choicerank Function). *Given a search system $\mathcal{S}$, a goal description Goal and a set of query modifications $\mathcal{Q}$, then we define the function choicerank as a function from $\mathcal{Q}$ to the interval $[0, 1]$ of real numbers.*

The ranking function might consult a number of independent properties like the quality of a query term as a discriminating term, e.g. a query refinement term should reduce the set of retrieved results significantly but must not result in an empty answer set etc.

There are a number of such issues involved, some of which are interlinked. An optimal ranking function will involve the use of some heuristically acquired parameters that reflect the particularities of the selected document collection. Part of that is encoded in the domain model.

Here we will concentrate on some obvious observations that guide the selection and ranking of query modifications based on the automatically acquired domain model. For simplicity we will only look at modifications of the actual set of query terms and ignore other document properties and system properties altogether. Therefore we will talk about queries instead of using the more precise term of goal descriptions in the following discussion.

We assume that a domain model has been constructed and the sample concept hierarchy depicted in Fig. 5.5 is part of this model. This hierarchy is only one among many others. Each node consists of a set of concepts. In this example we use letters, each of which stands for a concept. See Sect. 3.5 for a more concrete example from one of the sample domains.

What follows are a number of examples to highlight how this sample hierarchy can be utilized in building query modification options that are presented

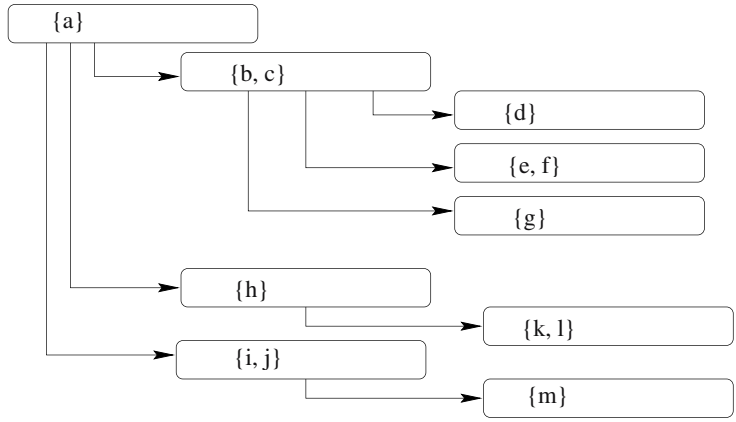

Fig. 5.5. Sample term hierarchy

to the user. Remember that each concept hierarchy is derived from the data found in the document collection. Therefore, an isolated node somewhere in the hierarchy is not very meaningful. The hierarchy needs to be interpreted top down: the root node represents a query. Each branch from the root down to the next level represents a query refinement, the refining query terms being located in each of the nodes in this next level (see Sect. 3.5). As a result of this we will only consult parts of the domain model where the query terms are close to the root node or where a coherent path of query terms can be constructed from the root down to a lower level.

The examples follow very closely the strategy of calculating query modifications applied in *UKSearch*. Other applications may apply different strategies.

Example 1

Suppose the user query consists of the single query term *a*. The domain model will be consulted to construct potential *query modifications*. The hierarchy in Fig. 5.5 only allows us to choose potential *query refinements* but not *query relaxations* for this particular query (an example where the hierarchy can be used for relaxations will be discussed further down). There are three refinements we consider. Each of them involves the addition of terms to the original query, so that the user will have the choice among the following three options:

- add terms *b* and *c* - or alternatively just one of these two - to the current query (as a result we now have a query that consists of query terms *a*, *b* and *c*), or
- add term *h* to the current query, or
- add terms *i* and *j* to the current query.

The weights of the corresponding arcs (not displayed in the figure) will be used to rank the modifications accordingly. The same is true in the following examples and will not be elaborated further.

Note that we do not go beyond a single refinement step (as explained earlier). This is the reason why we do not use the hierarchy to investigate any query modifications other than the ones listed above.

Note also that in this example we only outlined the query modifications that can be constructed based on the single hierarchy in Fig. 5.5. A simple dialogue system will indeed only consider hierarchies whose root node contains the query term *a*. However, the domain model typically consists of a large number of hierarchies. Apart from the example hierarchy one may consult others to construct potentially useful query modifications (e.g. hierarchies in which the query term is in a daughter node of the root, as we will see in the next example). Eventually, all the constructed choices need to be ranked and only the most highly ranked ones are chosen. For simplicity we ignore this here.

Example 2

Suppose the user query consists of the single query term *b*. First of all we would consult the concept hierarchy that has *b* in the root node and construct possible query refinements in the same fashion as in the first example. Apart from those query modifications we could try to utilize the structure in Fig. 5.5 to explore more options. The tree in Fig. 5.5 could be used to construct one query relaxation:

- replace query terms *b* by term *a* (leading to a new query that contains the single term *a*).

The example hierarchy is not suitable to infer any refinement options in this case. As in the first example, one needs to collect all possible query modifications, rank them and only select the most relevant ones which are then presented to the user to choose from.

Example 3

Suppose the user query is a conjunction of query terms *a* and *b*. The query refinement options to be considered would be:

- add term *d* to the current query, or
- add terms *e* and *f* to the current query, or
- add term *g* to the current query.

The query relaxation option to be considered would be:

- replace the current query by query term *a*.

Obviously, we could also offer the option to replace the current query by query term *b*. However, if we stick to the example hierarchy in Fig. 5.5, we only consider the replacement option listed above.

Example 4

Suppose the user query is a conjunction of query terms *h* and *b*. The query relaxation option to be considered would be:

- replace the current query by query term *a*.

The example hierarchy is not suitable to infer any refinement options in this case.

Example 5

Suppose the user query consists of the single query term *g*. The example hierarchy is not suitable for relaxation nor refinement. One might consider a refinement step that involves adding terms *a, b* and *c* to the query. However, that violates the idea of a *single* modification step.

5.6.4 Dialogue Strategies

The way we defined the dialogue aimed at maximum flexibility when it comes to implementing particular applications. A number of different dialogue strategies can be encoded using the same framework.

Some factors that influence the strategy being used are efficiency, size of the domain model(s), size of the database etc. For example, a very small document collection might make it necessary to investigate a large number of potential relaxation or refinement options for each query, while in larger collections this search space might have to be reduced significantly in order to generate quick answers. In the applications we use a strategy that follows closely the examples we have seen so far.

5.6.5 Customization

It is important to note that dialogue manager, domain model and the search engine can be completely independent components. The search engine can be locally installed or it can be called remotely. It is called using the query calculated by the *currentquery* function applied to the current goal description. The *Result* argument that is part of a dialogue state contains the results returned by the search engine. If the domain of interest (i.e. the document collection to be searched) is a local Web site accessible from outside like a university Web site, it might be appropriate to construct a call to some external search engine and evaluate the returned results (as long as the search engine

has sufficient coverage of the site and allows search on a particular Web site only, a feature that *Google* offers). On the other hand, searching a product catalogue or some other collection that is not available for Web search engines usually involves a specialized search engine that is more closely coupled with the dialogue manager (as it is the case in the YPA system).

Throughout the last few chapters we have been making references to the actual applications. Now is the time to introduce these implemented prototypes. The following part of the book will describe how some specific document collections can be processed into automatically constructed domain models and how the dialogue framework introduced in this chapter can be applied. This part will also discuss detailed evaluations of the prototypes.

We will first introduce *UKSearch*, a search system for Web documents. Two domains will be discussed, the *University of Essex* Web site as well as the *BBC News* Web site (Chaps. 6 and 7).

We will then look at the YPA, a search system for data in classified directories such as the *Yellow Pages* (Chap. 8).

Part II

Practical Applications

6

UKSearch - Intelligent Web Search

Finding information on the Web is normally a straightforward task. For most user requests the information can be located by applying a standard search engine using simple pattern matching techniques. However, by restricting the search to some smaller document collection (one that is still too large to be searched without appropriate tools) this can become a tedious task. Examples of such collections are corporate intranets or university Web sites. Typically a search will return large numbers of matching documents even in smaller document collections. If no matching document can be found, the user is usually either left alone with a great number of partially matching documents or with no results at all. These are well known problems and approaches for more sophisticated search systems exist to overcome them (see Chap. 2). But those approaches tend to rely very much on a given document structure or expensively created concept hierarchies. While this is appropriate for fairly well structured domains such as product catalogues and other applications where the information is stored in database formats, it is no help if the document collection is heterogeneous.

Surprisingly perhaps, the problem of not finding any document in the collection for a user query (a form of "data sparsity") is not necessarily a major problem in small domains. The log files of the search engine installed at the University of Essex Web site prove that the majority of queries that users submit result in a large number of matching documents despite the fairly small size of the collection. But unlike in general Web search where scalability issues prevent the application of more sophisticated indexing steps, we can build domain-specific concept hierarchies easily and rapidly in such well-defined document collections using the techniques introduced in the earlier chapters. These automatically created knowledge sources reflect the relations between documents or terms within those documents simply based on the available data.

Apart from that, collections of Web pages are well suited to verify the techniques introduced in this book, as these documents are typically marked up using HTML tags. This type of markup mixes visual markup and semantic

representation (as found in the meta tags for example). We turn this implicit knowledge into explicit relations.

The earlier chapters presented the conceptual framework. Here we discuss the practical steps that lead to an explicitly structured representation of a Web document collection. Frequently used HTML tags are used to define markup contexts (the fundamental units to extract concepts which are then arranged in a domain model).

The structure imposed on the data collection is employed in a dialogue system which assists the user with handling those queries that do not retrieve documents or result in large numbers of matches. We will see how the general dialogue manager introduced earlier is set up to work with the data collections discussed in this chapter.

We will however not focus on the links between concepts and individual documents or directories. The more interesting aspect is the construction of domain models that are not closely tied to the individual documents, mainly because a separable domain model is more flexible. The reason is that despite the ever-changing nature of a collection of Web documents we will not need to constantly update the model. A domain model that is not linked to the individual documents will still be usable once the document collection has been updated. It can simply be plugged into a search system.

In this chapter we will show how the extraction and search techniques are applied to two domains of Web pages[1], namely:

- the University of Essex Web site[2]
- the BBC News Web site[3].

Detailed evaluation results and discussions will be presented in the following chapter.

Finally, Chap. 8 will discuss an application where the focus is more on the dialogue than on the construction of domain models. This application, the YPA directory enquiry system, accesses a document collection that consists of classified advertisements.

6.1 Indexing Web Pages

As we have seen, classifications have been built, some of them impressively large, and they do work fine in either domain-independent context with lots of data to index (such as the Web) or in specialized applications with manually tailored classifications. But the semantic content of a document collection

[1] *Portions reprinted, with permission, from U. Kruschwitz. An Adaptable Search Systems for Collections of Partially Structured Documents. IEEE Intelligent Systems, 18(4):44-52, July/August 2003, and from U. Kruschwitz. Automatically Acquired Domain Knowledge for ad hoc Search: Evaluation Results. In Proceedings of NLP-KE'03, pages 525-532, 2003.* © *2003 IEEE.*

[2] `http://www.essex.ac.uk/`

[3] `http://news.bbc.co.uk/`

can vary; it may be very domain-dependent with no classifications ready to hand. It may change over time, and the amount of data may not allow a reliable classification. In these cases a structure constructed from the actual data seems more desirable.

We can identify a variety of structures in a collection of Web documents that could serve as a basis for extracting concepts and building a domain model as outlined earlier: internal document structure (HTML tags), link structure (hyperlinks), directory structure (the way documents are stored). We only use the internal document structure.

For Web pages we distinguish six types of frequently used markup contexts where candidate keywords can occur:

- *meta* information
- document *headings*
- document *titles*
- *anchor* text
- *emphasized* parts of the document
- any other text in the document.

Each of the (first five) contexts is identified by corresponding HTML tags. The details will be discussed later. Concepts are extracted based on just the first five contexts. This means that we ignore free text completely which reduces the size of the index tables significantly.

As a reminder, it should be emphasized that using meta tags only to identify concepts seems like a sensible alternative to our approach assuming the documents are in HTML format. For a classification task it was found that documents can be classified most accurately entirely based on meta tags [121]. However, only 26% of all documents in the Essex domain contain these tags, a figure similar to the one reported in [121].

Once the concepts have been selected, the construction of the domain models works as outlined in Sect. 3.5 (that section also gives examples of concept hierarchies for the Essex domain).

This all sounds like a very simplistic approach. But this simplicity allows us to rapidly construct new domain models and apply them without the need of any customization. There is however scope for incorporating even more structural information. So before we discuss the actual *UKSearch* system in Sect. 6.2, we will look at the potential that the existing document structure offers in terms of dialogue-based search.

The fact that in the current implementation we ignore all structure other than those few markup contexts does not mean that other contextual information cannot be incorporated in the indexing and search process for the given scenario. For example, relations between documents (e.g. via hyperlinks) can be considered to be expressing a relation between the concepts found in each of those documents. Furthermore, we could build some classification structure by associating entire directories with concepts found in the documents that are stored in those directories. To illustrate this, we will pick some actual

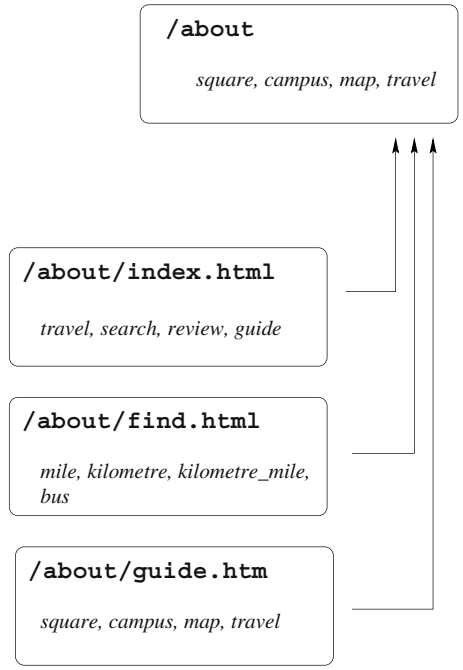

Fig. 6.1. Most frequent keywords

(though simplified) example using documents found on the Web server of the University of Essex.

A page entitled *About the University of Essex*[4] gives an overview of the University and has hyperlinks to two other files in the same directory: *Travel information*[5] and *Campus guide: finding your way around*[6]. Figure 6.1 shows what we obtain if we select the four most frequent keywords in the documents (ignoring stopwords which include the most frequently occurring nouns in the domain such as *university*, *colchester*, *essex*). The keywords selected for the whole directory (`/about`) are the most frequent keywords in all three documents.

This can be compared to the indexes that were selected applying our *concept approach* shown in Fig. 6.2. The figure indicates that the number of concepts we can extract for each document varies a lot (e.g. only a single concept could be identified in document `/about/index.html`). The terms associated

[4] `http://www.essex.ac.uk/about/index.html`

[5] `http://www.essex.ac.uk/about/find.html`

[6] `http://www.essex.ac.uk/about/guide.htm`

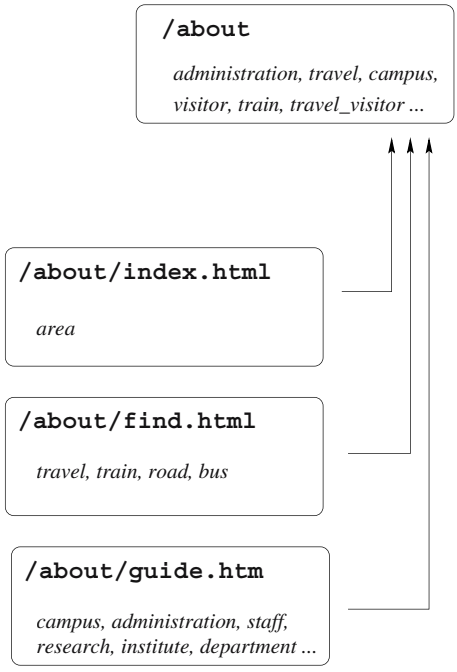

Fig. 6.2. Selected concepts

with the directory `/about` are the most frequent concepts selected for all the files listed in the directory.

Most strikingly, this demonstrates how frequent words are not necessarily the most desirable choices for a query system: the words *square* and *kilometre* are two examples which we rather do not want to see identified as significant terms (at least in this context). But we cannot simply add these terms to a list of stopwords to avoid them being selected, because they are not frequent enough in the document collection.

As indicated earlier, we do not use all of the available link structure in the prototype, and the example merely shows what additional structure could be incorporated in the knowledge extraction process. The YPA system discussed in Chap. 8 incorporates more structural information than the *UKSearch* system.

6.2 The UKSearch System

UKSearch[7] is an implemented prototype of a dialogue-based search engine that assists a user searching a Web collection. Its architecture is inspired by a number of observations. One of them is the fact that sophisticated search engines like *Google* have proven to work well on huge amounts of data. But even for smaller domains this technology will nicely handle a large proportion of the user queries. In the Essex domain a user who wants information about *"Computer Science"* will most likely be happy with the homepage of the Department of Computer Science, even though there are hundreds of pages matching the query; or to use the *"union"* example yet again, the most relevant matches (information about the *students union* in our domain) will probably be ranked highest in a list of matches if a good standard search engine is being used. In both examples the user will not be interested in entering a dialogue that constrains the query. It would be best to just display the results of the standard search engine.

To summarize, the main observations that influenced the design of *UKSearch* are:

- Sophisticated search engines without any dialogue component are sufficient for a large number of queries.
- Queries submitted to a Web search engine are usually very short (less than two words on average in the Essex sample domain).
- The majority of queries results in a large set of matching documents even in small domains.

The last two of those observations strongly support the use of a dialogue component. The very first point suggests the use of a standard search engine.

As a consequence, our approach does not aim at abandoning established search technology but at dealing with the remaining percentage of queries that cannot be handled in the straightforward fashion just outlined. The solution is sketched in Fig. 6.3. We combine two systems by passing the user query to the standard search engine (which does not have to be installed locally) and to the *UKSearch* system in parallel and merge both results (in fact, *UKSearch* calls the search engine through a defined interface). As soon as the standard search is finished, the top ranked results can be displayed. *UKSearch* runs the query against the domain model, and the dialogue manager constructs potential choices which are then presented alongside the matches returned by the search engine. In the *"union"* example that would mean that most user requests can be satisfied by a single search engine call. But if the expected documents are not among the most highly ranked ones, the user can select a potential choice (e.g. pick a query refinement term) to constrain the query (e.g. select *trade_union*). Obviously, we will only incorporate an external rather

[7] *UKSearch is short for "Udo Kruschwitz Search" and has nothing to do with other systems bearing the same name.*

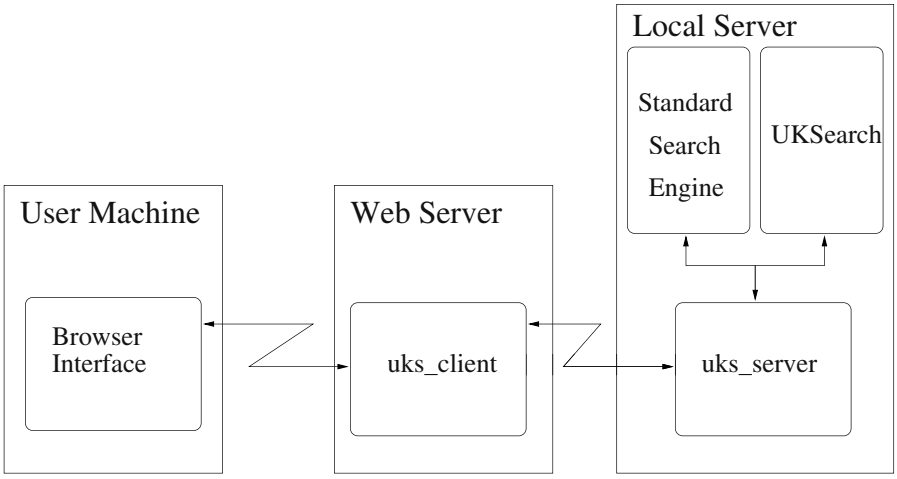

Fig. 6.3. Sketch of information flow in *UKSearch*

than a locally installed search engine if this search engine indexes more or less the same set of local documents that we use to build the domain model. For a publicly accessible Web site that should certainly be the case. If we are looking at an intranet, then we would consider installing a standard search engine locally.

The outlined system has a number of significant advantages over alternative ways of presenting choices for query modification. First of all, the initial response time is determined by the search engine technology that has been chosen. Secondly, the exploitation of structure found in the documents will only be needed if the standard search results are not satisfactory. Thirdly, this approach is significantly different from existing search engines that either use hand-crafted knowledge sources or rely on clustering methods on the fly.

The main component of *UKSearch* is an instance of the dialogue manager defined in Sect. 5.6. It is domain-independent, but is equipped with a set of interfaces to access:

- a domain model as defined in Sect. 3.5
- a standard search engine
- additional knowledge sources such as *WordNet*.

On top of the dialogue manager sits a graphical user interface for the interaction between user and system via a Web browser.

The customization for the two sample domains is identical apart from minor differences which are specified in the appropriate sections.

6.2.1 Indexing and Model Construction

The actual implementation of the indexing and model construction steps for the two sample domains are described here. Some implementational issues have already been outlined in Sect. 3.7. We define five different markup contexts: *Meta*, *Headings*, *Titles*, *Anchor*, *Bold*. The raw text for each of the specified markup contexts is selected at crawl time based on appropriate HTML tags. All tags are turned into a normalized form (small letters) prior to the extraction process. The text snippets that are then passed on to the indexing components are identified as follows:

- *Meta*: all text that is supplied as a value to the `content` attribute in the `<meta>` tags, given the `content` attribute is accompanied by an attribute `name` or `http-equiv` and the corresponding value matches the pattern `/keyword/` or `/description/`
- *Headings*: all text that is marked using one of the heading tags (`<h1> ... <h6>`)
- *Titles*: all text that is marked using the `<title>` tag
- *Anchor*: all text that is marked using the `<a>` tag
- *Bold*: all text that is marked using the `<b>`, `<i>`, `<u>`, `<strong>`, `<big>` tags.

The indexing step processes the text according to the guidelines listed in Sect. 3.7, and each index term is associated with the document it was found in. An alternative approach would be to associate the *anchor* text with the document that the anchor points at. Two other possible modifications are to record font changes as some sort of markup (e.g. using the `<font>` tags) and to distinguish more markup contexts by not collapsing different HTML tags into a single context (e.g. see the definition of the *Bold* context). Doing this one may well distinguish 10 or 20 different contexts.

Two types of concepts are extracted from the index database: concepts of type 2 (type-2 concepts) and concepts of type 3 (type-3 concepts). Remember that type-2 concepts are those index terms that have been found in at least 2 different markup contexts within some document. By definition, any type-3 concept is also a type-2 concept.

The next step is the automatic construction of a domain model. We build a single model that incorporates both type-2 and type-3 concepts using a *back-off* strategy that always consults type-3 concepts before backing off to type-2 concepts if necessary. Section 3.5 contains a detailed description of the construction algorithm. The implementation modifies that process as follows:

- We create a root node for each concept of type 2 and type 3. This implies we construct as many hierarchies as there are concepts, although these hierarchies may turn out to be trivial, i.e. contain a root node only.
- We restrict the maximum number of branches in each node to a fixed number max. If there are more than max branches leaving a node, we

select those max daughter nodes whose corresponding queries match the highest number of documents. We use $max = 20$.

- We restrict the minimum number of branches in each node to a fixed number min. If there are less than min branches leaving a node, we discard all of those branches. We build models using $min = 5$. This needs some explanation. The question is, why do we use a minimum threshold at all? The answer is that we would like to be able to suggest more than just one or two query refinement options if we can suggest any at all. In fact, we only require a minimum number of branches to be constructed as long as we still have some back-off options. If we end up in the very last stage of this back-off construction process, then we drop the requirement of a minimum number of branches.
- The back-off approach is used in step 2 of the algorithm in Sect. 3.5. For each node in the hierarchy we try to construct branches according to the algorithm using the concept type $n = 3$, i.e. every paths from the root to each of the daughter nodes of the current node are constructed based on type-3 concepts. If this method results in less than min branches leaving the node, we discard them and back off to concept type $n = 2$. This back-off step does not affect any of the branches created in earlier steps.
- The model construction process is driven by matching queries against the document collection. We did not specify how we would determine whether a document matches a query represented by a set of concepts. Here we apply a straightforward matching function: a document matches a query if all the query terms were found to be concepts in the document. In other words, the matching function $match$ that matches sets of concepts C (i.e. a query) to documents is defined as follows:

$$match(C) = \{d | \forall c \in C \; concept(c, d)\}$$

- Each arc in a concept hierarchy is to be assigned a weight. This weight will allow us to rank options presented to the user. In the model construction process this weight allows us to decide which branches should be encoded in the hierarchy and which ones should be discarded. The underlying assumption in our construction process is to include more frequent in favour of less frequent concepts in the domain model. We therefore rank branches in a concept hierarchy according to the number of matches each of the nodes represents. The weight for each arc is then calculated as a ratio $\frac{|match(C_1)|}{|match(C_2)|}$ where C_1 is the query representing the daughter node and C_2 is the query representing the mother node. So if we have to consider a large number of possible daughter nodes, we will discard all those potential nodes that would lead to very few matches (i.e. very specific queries) since we restrict the number of branches leaving a node to $max = 20$.
- One more thing: we consider noun phrases to be much better in conveying meaning to the user than single nouns, which is why the back-off strategy involves another sub-step. We first try to create branches for noun

phrases of a particular concept type (applying the same thresholds *max* and *min*), otherwise we consult all concepts of that type. The idea of using compounds (noun compounds in particular) as potential query refinement terms in ad hoc search tasks is not new. Anick gives a detailed motivation (including references to related work) for his work on context-based information retrieval which adopts "the noun compound as the primary unit of interactive terminological feedback" [8].

The thresholds *max* and *min* were chosen in the light of the application: these values represent the number of refinement choices to be supplied by the domain model for a given query.

We also restrict the depth of the domain models to a maximum path length of 2. The reasons are twofold. One reason is efficiency, i.e. the construction of a bigger model requires substantially more time and resources. But the more intuitive reason is that based on the user studies referenced in the chapter on related work we expect that a user will typically not navigate further down than two levels. And even if that was the case, then the small set of branches on the next level can quickly be constructed on the fly applying the same techniques, assuming that the system has access to the appropriate index tables.

For the BBC domain we treat the most frequently occurring concepts as stopwords. This excludes about 100 very frequent concepts. For the Essex domain we do not define any such stopwords.

6.2.2 Dialogue Strategy

UKSearch's dialogue manager is an implementation of the system outlined in Sect. 5.6. Here we demonstrate how each of the parts of the dialogue manager is set up.

Properties

Due to the choice of domain model and system setup we have only a single *document property* called *keyword* that typically takes a set of keywords as its value. We say "typically", because apart from representing a conjunction of query terms we can also represent disjunctions of conjunctions (see the example further down).

Since we do not allow the user to set any system parameters and because the search engine and domain model are completely separate components we do not consider any *system properties*.

That leads to an extremely simple *goal description* which consists of a single attribute-value pair that expresses that a document matches a set of terms. An example representation of the user query *"union"* as a goal description *Goal* would be:

```
Goal = [keyword({union})]
```

The value of the *keyword* attribute is updated accordingly in query modification steps by adding, deleting or replacing terms. For example, a user might have selected one of the options proposed by *UKSearch* such as *"Add the term trade_union to the query"*. The new goal description would then look like this:

```
Goal = [keyword({union, trade_union})]
```

If however, we want to represent a disjunction such as *"union OR european_union_representative"*, then the corresponding goal description looks as follows:

```
Goal = [keyword({{union}, {european_union_representative}})]
```

For the representation we assume a normalized format. Any query term is transformed into small letters. Typographical characters are deleted. Double quotes in the user input are not deleted. They indicate that a sequence of words should be treated as a phrase rather than a list of individual words. This idea follows the convention of the use of double quotes in standard search engines.

The default search engine in *UKSearch* is Google which provides an API[8] to incorporate Google's results in applications. However, any other search engine that has access to the document collection and that provides an interface could be used. The translation of the goal description into a user query (i.e. a Google request) is a straightforward processing step that follows the input format required by Google's API.

Dialogue Setup

The *input* is either user typed text or a selection of a tick box that is associated with a potential choice offered by the dialogue system.

The *dialogue history* contains the last query. This structure is currently not exploited in the *UKSearch* system. It may well be used in the future. It could be applied to avoid presenting exactly the same query modification options twice in a search task.

The *currentquery* function turns a goal description into a search engine call that contains all the keywords as a flat list of query terms (this list may however contain conjunctions or disjunctions). Such a search engine call encodes specific information about the domain to be searched in case the search engine is not locally installed.

The *retrieve* function matches a query to the results returned by Google (we treat these results as a ranked list of matches).

The only notable dialogue state is the *Display* state. The dialogue manager moves into one of two other states (*Start* state or *Meta* state), if the user requests to restart the dialogue or asks for help. The input parser detects such requests (based on pattern matching) and passes this information to the dialogue manager.

[8] `http://www.google.com/apis`

Dialogue Function

The dialogue function *trans* which maps a dialogue state and the user input to a new dialogue state is very simple:

- If the user *types* an input, then the goal description is updated accordingly by adding the new input term(s) to the query term(s) already encoded. The new dialogue state is calculated by selecting potential choices for the new goal description (see below) and applying the necessary query and retrieve functions.
  However, if the input parser interprets the input as a "correction", then the goal description is updated by substituting the new input term(s) for the query term(s) currently held in the goal description. An example of such a user input is *"No, I want the accommodation office"*.
- If the user *selects* a potential choice $< Goal, Hist, Input >$, then *Goal* is the new goal description. The rest of the new dialogue state is calculated as above. Note that the user just ticks a box/hyperlink, but the system interprets this as if the user has performed some query modification encoded by the value of $Input$ in the potential choice.

If the input parser detects a request for help (e.g. *"Help"*), then the dialogue will move into the *Meta* state. The goal description remains unchanged, but the user will be given more detailed feedback about the options currently available. Any input in the *Meta* state is handled in the same way as it would have been handled before moving into this state.

Calculation of Potential Choices

Whenever potential choices need to be constructed (i.e. whenever the dialogue function is called), we perform these steps:

- Calculate query refinements
- Calculate query relaxations
- Rank all query modifications
- Select the highest ranked query modifications and construct potential choices.

Since we have uncoupled the domain model from the actual database of documents, the dialogue manager does not actually keep track of what the search engine returns. In other words, in any case we calculate relaxation as well as refinement options.

We apply the domain model to explore a fairly restricted space of query modifications. This is because the domain model is custom-built for exactly this process, i.e. finding refinement or relaxation terms for a given query. There is no need to go deep into a hierarchy, nor are we interested in exploring sets of nodes that are not immediate neighbours in such a hierarchy.

We shall first outline how refinements and relaxations are constructed based on the domain model. We will then describe this process more formally.

- Query refinements are calculated by proposing a single concept that could be added to the current query. Since the domain model hierarchies represent (hypothetical) query refinement steps, we just have to find a coherent path that takes us from the root node further down in a hierarchy, and on that way we collect all the current query terms (and do not skip any nodes). Following the last node on this path we have the nodes that contain query refinement terms.
- Query relaxations are constructed by either breaking the current query into parts (i.e. by deleting query terms) or by substituting the query by a single concept. This single concept has to be found in the root node of a concept hierarchy, and all query terms are found in the direct daughter nodes of that root node.

To express that more explicitly, we assume the current goal description is represented by a set of concepts C. Then we construct query modifications as follows:

- Query refinements are constructed by the addition of a single concept r to the current query for every hierarchy h with root node *root* so that r is a *potential query refinement term* for C in h (i.e. there is a node n such that $path(root, n, h)$ with $r \in n$ and for every $c_1 \in C$ there is a node n_1 on this path such that $c_1 \in n_1$) and we also require that for every node n_2 on this path there is a $c_2 \in C$ such that $c_2 \in n_2$.
- Query relaxations are constructed by replacing the current query by a single concept r for every hierarchy h with root node *root* so that r is a *potential query relaxation term* for C in h and:

$$r \in root \ \wedge \forall c \in C \ \exists n \ c \in n \wedge path(root, n, h) \wedge |path(root, n, h)| = 1$$

  Every goal description represented by a *single query term* is also treated as a potential query relaxation (see the example in Fig. 1.5).

The definitions of *potential query refinement term* and *potential query relaxation term* (Definitions 3.10 and 3.12 in Sect. 3.6) are more general than the conditions outlined here. Basically, we only consider concepts that are most closely related to the set of query terms. Furthermore, we treat each node in the domain model as if it only contained one concept (we pick the first one). Alternatively, we could add all concepts found in a node as depicted in Fig. 6.9 (discussed further down).

The function *choicerank* ranks all query modifications that have been considered as potential choices. The highest ranked choices are presented to the user (we use a maximum number $max = 20$).

The actual function uses the weights found in the concept hierarchies. The aim is to select those options that retrieve the highest number of matches. Note that these numbers are based on the data set that was used to build the model and that this does not involve any database calls.

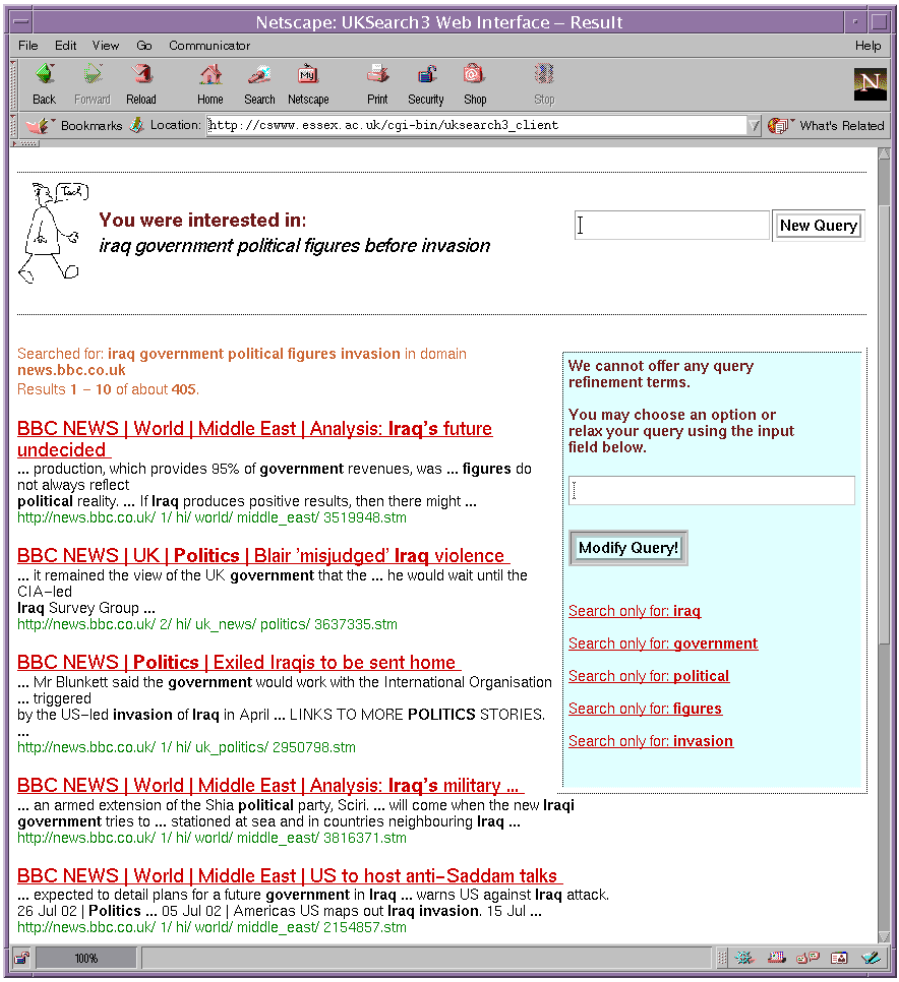

Fig. 6.4. *UKSearch*: query relaxation

Our ranking function ignores all query relaxations if there are potential query refinements. This is based on the observation that *too* general queries are much more likely than *too* specific queries (to be discussed as part of the log analysis in Sect. 7.1). An example of a query relaxation step initiated by the system in which the query is simply broken into individual terms can be seen in Fig. 6.4 where a user asked for *"iraq government political figures before invasion"* in the BBC News domain.

If the domain model cannot deliver any refinement options, then we imitate the domain model construction process with the query terms dynamically. In other words, we try to locate concepts that are related to all query terms and construct the most highly ranked choices. The motivation here is that

there may as well be sensible options which have not found their way into the domain model. Remember that we try to keep the size of the model manageable. The trade-off of not calculating all possible links in the model is that certain branches will not turn up in the domain model at all. Those can however be constructed on the fly as long as *UKSearch* has access to the index tables needed for the model construction.

In addition to that we also filter the new terms that will be used for query modifications in a way that only the longest possible query terms are considered and all substrings are ignored. We want to avoid a user having to choose between *chancellor_gerhard_schröder* and *gerhard_schröder*. We discard the second one if both terms were found to be possible refinement options. This seems a sensible strategy although a query for *"dow"* in our BBC News domain will then not include a refinement term *dow_jones* which is discarded in favour of *dow_jones_milestones*.

The potential choices are then passed to the dialogue manager to be encoded in the new dialogue state. It has already been mentioned that the user just has to tick a box (or a hyperlink), and behind the scenes the query is updated accordingly. On the screen the user will find information about what type of query refinement a box represents.

A final word about query refinement terms: a user may choose to add such a term to the original query, but a user may also choose to replace the entire query by the suggested term (similar to *AltaVista*'s *Prisma* refinement tool [7]). This is to make the search system more user-friendly and to allow the user to explore the document collection rather than strictly narrow down a particular search request.

6.3 Sample Domain 1: Essex University

The first domain that we have chosen for the *UKSearch* Web search system is the Web site of the University of Essex (Fig. 6.5 shows the starting page of the online search system). Apart from its size and accessibility there is one main reason that makes this domain a suitable one. Log files for the locally installed search engine have been made available which allowed us to perform technical evaluation steps based on real data.

Figure 6.6 is a screenshot of the *UKSearch* system that shows the most highly ranked potential choices presented to a user following a user query *"union"* (terms that can either be added to the query or replace the original query altogether). We have seen this screenshot before in the introductory chapter.

We want to present some statistics to demonstrate the amount of data extracted and indexed. In the University of Essex domain we currently index about 38,000 Web pages, which are all HTML or ASCII pages accessible from the starting page presuming the robot exclusion files do not prevent robots to crawl them. We exclude certain directories that contain enormous amounts

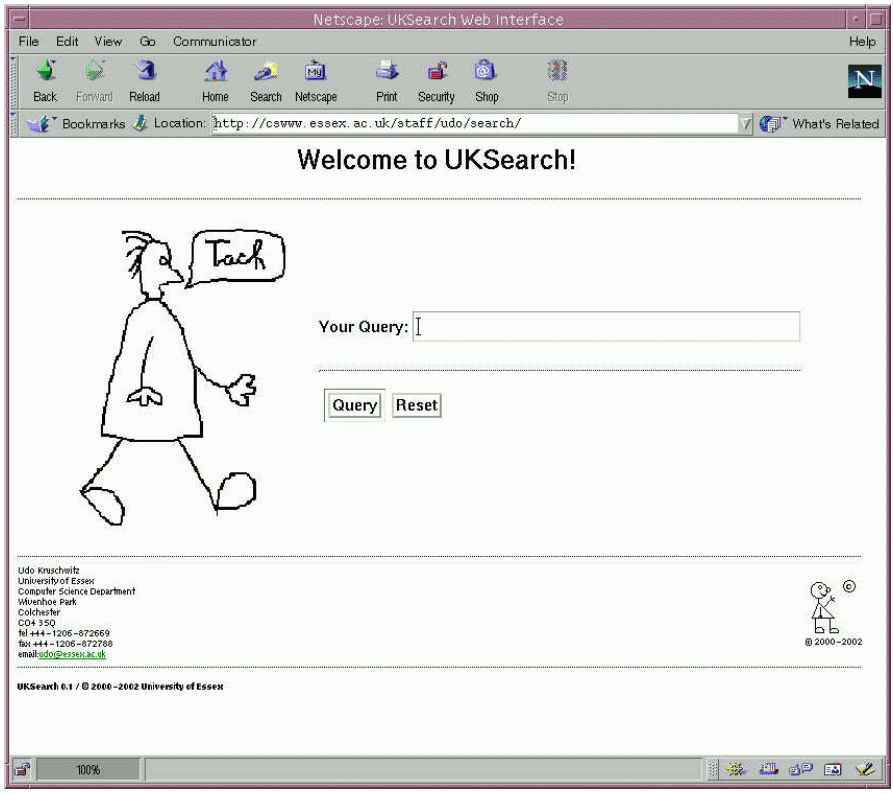

Fig. 6.5. *UKSearch*: user interface

of documentation (such as Java manuals). Those documents have restricted access rights and are not accessible from outside the university anyway.

6.3.1 Index Tables

Table 6.1 contains some statistics for the Essex domain.

Table 6.2 is an overview of how many concepts and keywords we find on average for a Web page in the sample domain. It also contains figures about what percentage of pages contains concepts at all.

We also want to give a list of some selected concepts (the first five in the alphabet and some other examples refered to in the text) that were found in the Web pages of our sample domain. Table 6.3 is a short summary. The table contains information for each one of the selected concepts about how many related concepts were found in total, in how many documents in the collection the term was identified as a concept, and (as a comparison) in

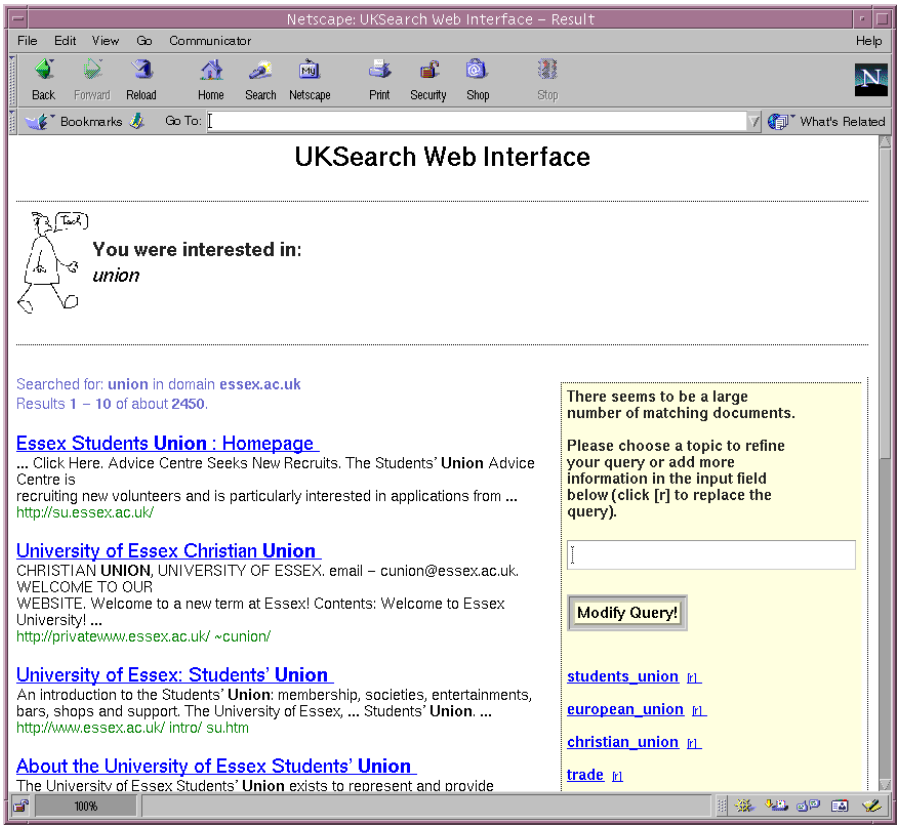

Fig. 6.6. *UKSearch*: system's response following a user query

how many documents this term was identified as a simple keyword, i.e. not necessarily as a concept.

6.3.2 Domain Model

Table 6.4 characterizes the domain model (remember the maximum depth of 2). Non-trivial models are models with more than one node (i.e. not just the root node). It may be surprising to find a fairly large number of trivial models. This does not mean that these models actually contain only a single concept in the root node (this would be a bit strange considering that there are not many concepts without any related concepts). However, it is common to find groups of related concepts in the root node (e.g. the example of concept *isabelle* that we have refered to a few times: there is a concept hierarchy that consists of a root node only, a root node that contains the concepts *isabelle*, *proof*, *theorem_prover* and *theorem*).

Table 6.1. Some statistics describing the *University of Essex* domain

	Number
Indexed Pages	37,561
Distinct keywords in *bold* context	85,960
Distinct keywords in *title* context	11,398
Distinct keywords in *anchor* context	56,042
Distinct keywords in *meta* context	6,725
Distinct keywords in *headings* context	31,141
Distinct type-3 concepts	3,205
Distinct type-2 concepts	14,587
Pairs of related type-3 concepts	39,314
Pairs of related type-2 concepts	2,385,944
Percentage of concepts with related type-3 concepts	92.1%
Percentage of concepts with related type-2 concepts	98.6%

Table 6.2. Page statistics of the *University of Essex* domain

Average number of keywords per page (in any of the five contexts)	71
Average number of type-3 concepts per page	1.6
Average number of type-2 concepts per page	6.8
Percentage of pages with type-3 concepts	46.1%
Percentage of pages with type-2 concepts	88.6%

Table 6.3. Concept examples in the *University of Essex* domain

Concept	**Related Concepts**	**Documents with concept**	**Documents with keyword**
aave	20	2	33
abbreviations	16	4	53
abdala	1	1	67
abdel	5	3	33
abdel_salhi	3	2	9
...	...	...	...
language	825	303	2,518
...	...	...	...
union	163	140	1,589
...	...	...	...

If we applied a different matching function in the domain model construction process - one that matches query terms against any keyword found in the documents instead of concept matches only - we would get far fewer trivial

Table 6.4. Domain model statistics for the *University of Essex* domain

Number of hierarchies	14,587
Average number of branches leaving root node (non-trivial models)	8.2
Average number of leaf nodes (non-trivial models)	37.4
Percentage of hierarchies with root node only	46%

hierarchies. It would however also mean that we had to deal with a much larger database in the model construction process.

If we only look at those hierarchies which have type-3 concepts as root nodes, we get the breakdown in Table 6.5.

Table 6.5. Domain model statistics for the *University of Essex* domain (root nodes are type-3 concepts)

Number of hierarchies	3,205
Average number of branches leaving root node (non-trivial models)	9.0
Average number of leaf nodes (non-trivial models)	44.8
Percentage of hierarchies with root node only	22%

6.3.3 Concepts *vs.* Real User Queries

In the introductory chapter we assumed that the concepts extracted in the domain model construction process are likely to be among those terms that users submit as real queries when they search the document collection. The log files of queries submitted to the University of Essex search engine prove that this is indeed a sensible assumption.

These log files - recording all queries over a period of several months - reveal that 74% of the top 100 most frequently submitted queries match our definition of a *concept* (i.e. there are documents which contain these terms in more than one markup context), either single terms or compounds. Note that the top 100 most frequently submitted queries make up about a quarter of the entire query corpus although they constitute less than 1% of all unique queries. For an average query (i.e. not just one of the 100 most frequent ones) we get a 41% chance that the query matches a concept. These figures are based on exact matches only. If we apply base form reduction, stemming or partial matching of queries against concepts the overlap is higher (for example, of all the unique single terms that are found in the top 100 queries, about 89% are actually concepts, and for the average query this figure is 72%).

For the same sample domain we also compared the terms that we identified as concepts with terms found in the meta tags of the documents (the markup environment that allows authors of HTML documents to provide keywords

and topics describing a document's content). We established that a query term is more likely to be extracted as a concept than listed as a keyword in the meta tag environment. The figure of 89% given above compares to 80% using meta tags, and the 72% chance of an average query term matching a concept compares to 58% matching a word in the meta tags in our domain. The significance of this observation is that a domain model which relies entirely on keywords placed by authors in the meta tags will be lacking a number of terms typically submitted in user queries.

6.4 Sample Domain 2: BBC News

The second domain that we have chosen is the BBC News Web site. The aim was to find a domain that is significantly different from the first sample domain without having to invest into additional resources or worry about language-specific tools (i.e. the part-of-speech tagger). The *UKSearch* system would then be run on the new domain without modification.

The BBC News site was found to be suitable because it is a non-academic site representing a much larger collection of documents which nevertheless can be processed using the existing tools. Most importantly, there is no restriction imposed to an automatic crawler (as stated in the robot exclusion file of the site). Furthermore, the terms and conditions for academic purposes permit the use of material for educational purposes.

It should be stressed, that this domain is particularly interesting since it contains breaking news articles alongside archived documents. That makes the document collection changing very fast and the content very domain-specific. It also means, that the domain models resulting from the collection are much more unpredictable than in our first sample domain. Whereas one could imagine to handcraft a domain model or some sort of ontology for the Essex Web site, it appears much more difficult to do that for the BBC News domain and make sure that this model is kept up-to-date. This is the type of document collection for which the model construction methods presented in this book seem most promising.

Using the same setup as for the Essex domain it is possible to build a domain model in exactly the same way.

Figure 6.7 displays the system's response to the user query *"union"* in this domain. This version of *UKSearch* provides two input fields, one to modify the current query and another one to start a new search.

Now have a look at Fig. 6.8 to get an idea of what sort of relations the domain model encodes in a news domain. We were interested in information about the political figures in Ukraine, but unfortunately had problems getting the names spelled right. So we queried for *"ukraine"* and can then just replace the query by the appropriate name.

Figure 6.9 is another example screenshot of a query submitted in this domain (here we use the simplified version of *UKSearch*). This is a good example

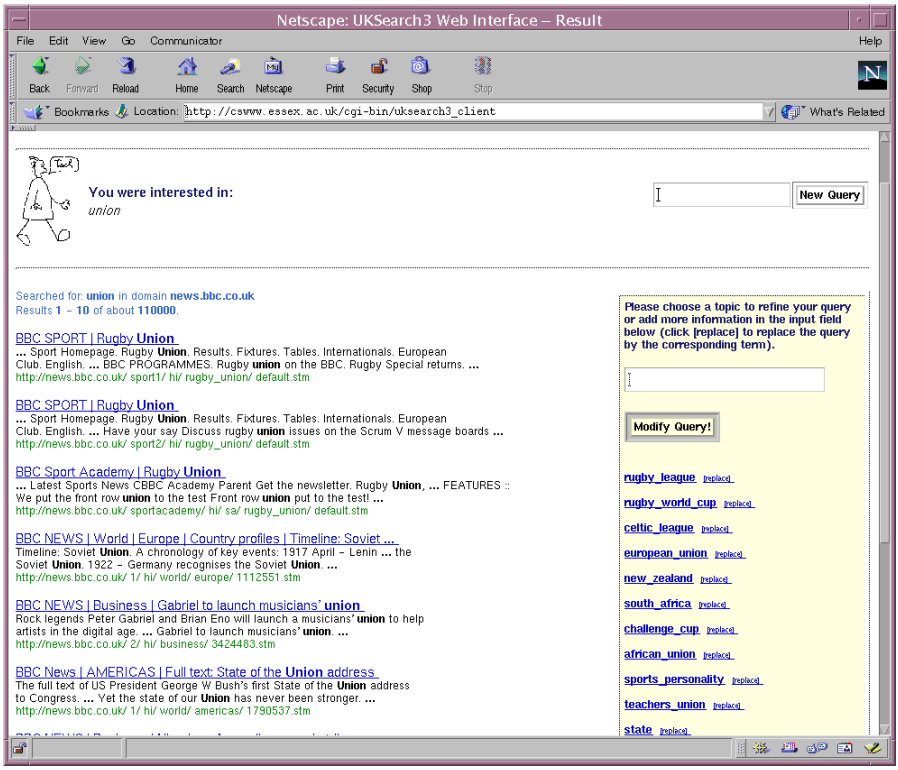

Fig. 6.7. *UKSearch*: system's response to the user query *"union"*

to explain how we can treat domain model nodes that contain more than one concept. If a refinement consists of more than a single term (i.e. the appropriate node in the hierarchy contains more than a single concept), then all terms could be added to the current query. Figure 6.9 is an example. The example query submitted by the user was *"concorde crash"*. The domain model construction process identified a concept *concorde_crash*. Therefore, *UKSearch* treats the query as a phrase. A hierarchy with concept *concorde_crash* in the root node was built in the offline construction process. The three branches with the highest weight connect the root with nodes that each contain a single concept. The fourth branch however connects the root with a node that contains three concepts. The explanation of this is that whenever the index term *concorde_crash* was found to be a concept in a document, then this document contained either all three related concepts *relatives*, *lawyers* and *concorde_cash* or none. There are a number of pages matching these concepts, one of them entitled *"Relatives accept Concorde cash"*, another one *"Lawyers back Concorde cash deal"*.

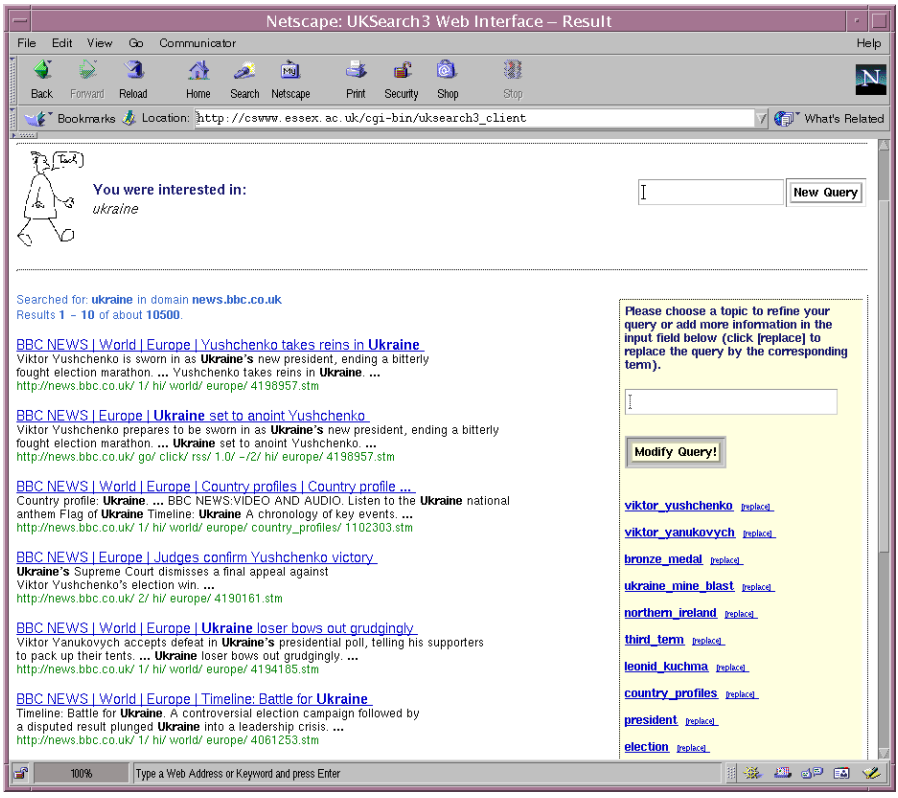

Fig. 6.8. *UKSearch*: system's response to the user query *"ukraine"*

Some notable differences detected in this domain compared to the Essex domain are:

- The structure of the document collection as well as the internal document structure is much more consistent throughout the collection. This is a fairly obvious observation.
- Meta tags are used much more frequently.
- Much less use of heading tags is being made, but significantly more text is encoded as anchor text.
- A comprehensive stopword list for this domain would include rubric names and names of correspondents, words that are very frequent in the lists of concepts.

Due to hardware restrictions we only collected the first 50,000 pages (of several hundred thousand in total) that have been found by starting at the top level page of this domain. The domain models were built based on this selection.

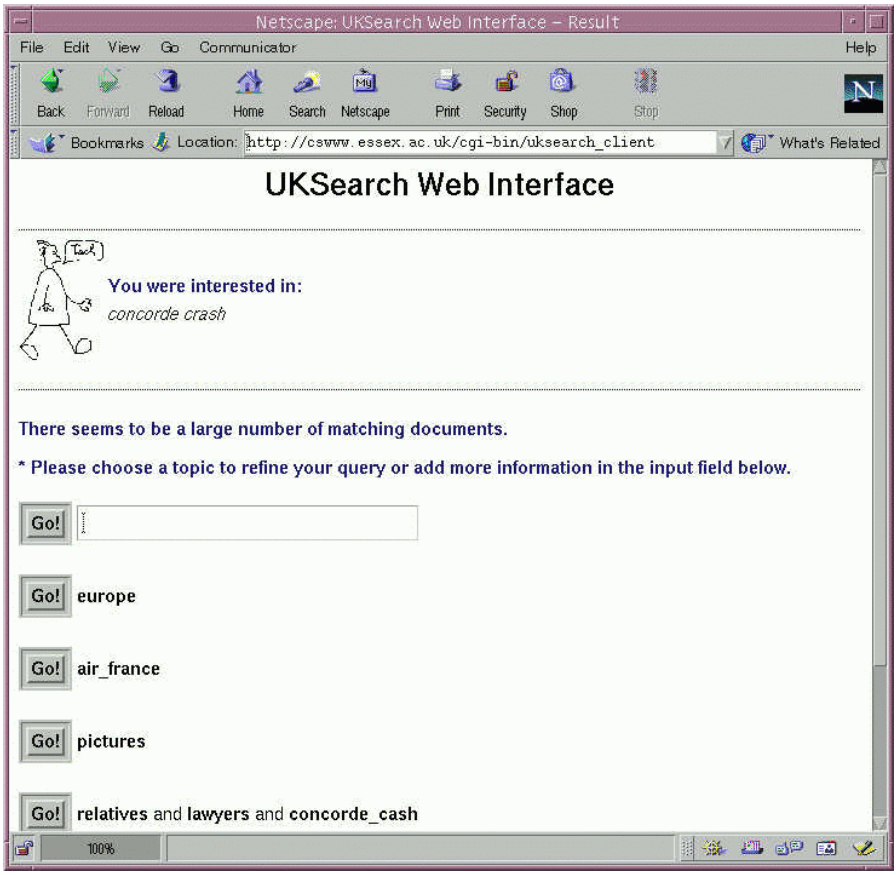

Fig. 6.9. *UKSearch*: query refinement using more than one concept

Note that the actual number of pages in the statistics is slightly lower than 50,000, because not all of the crawled pages turned out to be text (despite the exclusion of certain file types and file name patterns at crawl time).

6.4.1 Index Tables

Table 6.6 contains statistics for the BBC News domain. Most notable is the fact that there are almost no type-3 concepts (and only a single type-2 concept) which have no related concepts of the same type. Apart from that, there is generally much more markup used in this domain.

In terms of selected concepts and keywords for every single document we get the picture in Table 6.7. Note the differences compared to the Essex figures in Table 6.2.

Table 6.6. Some statistics describing the *BBC News* domain

	Number
Indexed Pages	48,535
Distinct keywords in *bold* context	101,445
Distinct keywords in *title* context	34,569
Distinct keywords in *anchor* context	130,656
Distinct keywords in *meta* context	58,408
Distinct keywords in *headings* context	8,991
Distinct type-3 concepts	13,138
Distinct type-2 concepts	34,391
Pairs of related type-3 concepts	273,776
Pairs of related type-2 concepts	2,302,828
Percentage of concepts with related type-3 concepts	99.6%
Percentage of concepts with related type-2 concepts	100%

Table 6.7. Page statistics of the *BBC News* domain

Average number of keywords per page (in any of the five contexts)	163
Average number of type-3 concepts per page	4.9
Average number of type-2 concepts per page	13.9
Percentage of pages with type-3 concepts	94.7%
Percentage of pages with type-2 concepts	99.5%

6.4.2 Domain Model

We also want to present some domain model statistics here so that we have a comparison between the domain models in each of our two sample domains (cf. the tables in Sect. 6.3 for the Essex domain). If we look at all the concepts in the model identified for the BBC News domain (i.e. type-2 and type-3), we get the figures in Table 6.8.

Table 6.9 tells us how those figures look like when we only consider those hierarchies which have type-3 concepts as root nodes.

Table 6.8. Domain model statistics for the *BBC News* domain

Number of hierarchies	34,391
Average number of branches leaving root node (non-trivial models)	4.6
Average number of leaf nodes (non-trivial models)	10.2
Percentage of hierarchies with root node only	55%

Table 6.9. Domain model statistics for the *BBC News* domain (root nodes are type-3 concepts)

Number of hierarchies	13,138
Average number of branches leaving root node (non-trivial models)	6.0
Average number of leaf nodes (non-trivial models)	15.0
Percentage of hierarchies with root node only	27%

6.4.3 Adjusted Dialogue Strategy

In our first document collection, the Essex Web site, we could assume that the domain model we constructed would be relatively stable. However, as we pointed out already the BBC News domain represents a rapidly changing collection. But if we just construct the model once and then apply it, we will end up with a rather static knowledge source that will be out-of-date after some time unless we re-run the construction process.

What we do is we adjust the dialogue strategy for this type of document collections by combining the query refinement suggestions derived from the domain model (as explained earlier) with query refinement terms extracted on the fly from the documents that match a user query, and those terms are presented alongside each other.

For the extraction of knowledge on the fly we use the titles and snippets of the best matching documents as they are returned by the search engine and process this text in the same way as we process the document collection prior to deriving the domain model. That means we assign parts of speech, select nouns and certain noun phrases. Finally we select the most frequent ones and mix them with the refinement terms suggested by the domain model. We display up to 10 terms derived from the domain model followed by the 10 most frequent ones calculated on the fly. The output looks exactly like what we have seen in the Essex domain, i.e. the output format does not differ from a system that only uses the domain model for the construction of query modification terms.

Figure 6.10 depicts the data flow in this system setup which is not dramatically different from what we saw earlier in Fig. 6.3 apart from the fact that query modification suggestions are now also derived from the best matching documents returned by the search engine following a user query.

6.5 Implementational Issues

We want to conclude this chapter with a discussion of some technical aspects of the *UKSearch* system.

It has already been pointed out that both domains that we discussed use basically the same setup, in particular the domain model construction process

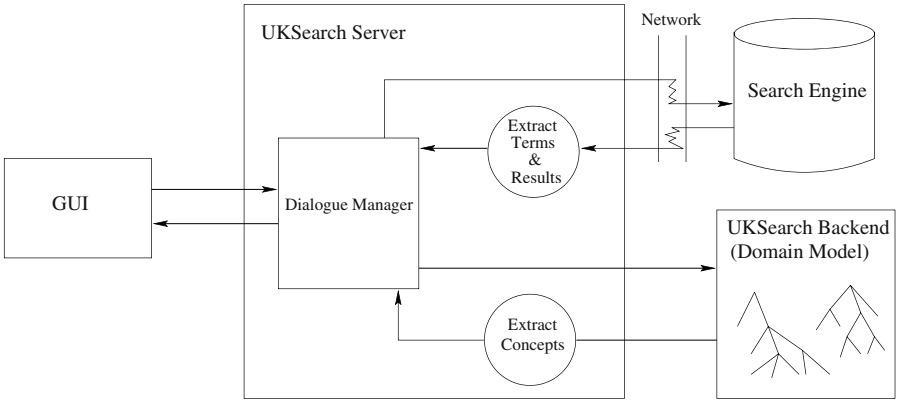

Fig. 6.10. Sketch of information flow in *UKSearch* (BBC News setup)

is almost identical. We also highlighted some differences in the dialogue strategy.

The gathering robot starts at some root point and collects in a breadth-first strategy all relevant Web pages that can be found. It currently ignores all documents that are not in HTML format or located in a different domain. The gathered documents are passed to the indexing component.

The robot as well as most of the index construction programming is done in Perl making use of existing modules (LWP, HTML, URI, Text etc.). For the indexing process we also use the Brill tagger [18].

The Porter stemmer [122] is an optional part of the system. The system administrator would have to decide whether it should be applied or not since the indexing and the query processing steps would ideally apply the same tools. Other resources would have to be applied for languages other than English. An anecdotal note about the stemmer: while there are a number of implementations of the Porter stemmer available, they all seem to interpret the algorithm somehow differently. Just one example is the word *artificial* which is supposed to be stemmed to *artifici*. The Perl modules Text::English or Lingua::Stem return *artificy* and *artificial*, respectively. We decided to apply the C and Perl implementations provided by Porter himself.[9]

The Web-based *online* dialogue system runs as a *Sicstus* Prolog executable accessed via sockets. Perl scripts pass the user requests from the browser to the *Sicstus* system and return the answer once the system is finished. This setup permits the handling of multiple user requests which are pipelined. It is possible to run several instances of the *Sicstus* system which all access the same database. In that case the Perl script responsible for contacting the *Sicstus* system will also work as a multiplexer. The external databases we are using are *mSQL* and *Oracle*.

[9] `http://www.tartarus.org/~martin/PorterStemmer/`

There is now an alternative implementation which is purely *Java*-based. The generic interfaces in that new system allow access to *MySQL* as a database system and the open source search engine *Nutch*[10].

The system was initially in parts based on the YPA and subsequently reimplemented.

Our platform is a *Sun Blade* 150 with 640 MB working memory running Solaris 8. It performs each of the crawling, indexing, uploading and model construction steps in a matter of hours.

Our choice of sample domains was justified earlier. However, other than for the reasons given earlier these are arbitrary choices and the techniques should be similarly applicable to other domains. In fact, we did apply the same techniques to a number of other sample Web sites to verify that no manual modification (other than the definition of a start page and the domain name) would be necessary. Example sites that we indexed are the University of Edinburgh, the University of Sheffield and the University of Brighton sites.

An interesting aspect is that the domain models for the universities can differ quite dramatically. If we look at the screenshot in Fig. 6.11, we see that a user searching for *"language"* in the University of Brighton domain will get a large number of very technical query modification suggestions such as *object_oriented_language*, *common_lisp* and *assembly_language*. This is very different from the Essex domain which is heavily biased towards documents about language and linguistics.

[10] `http://www.nutch.org`

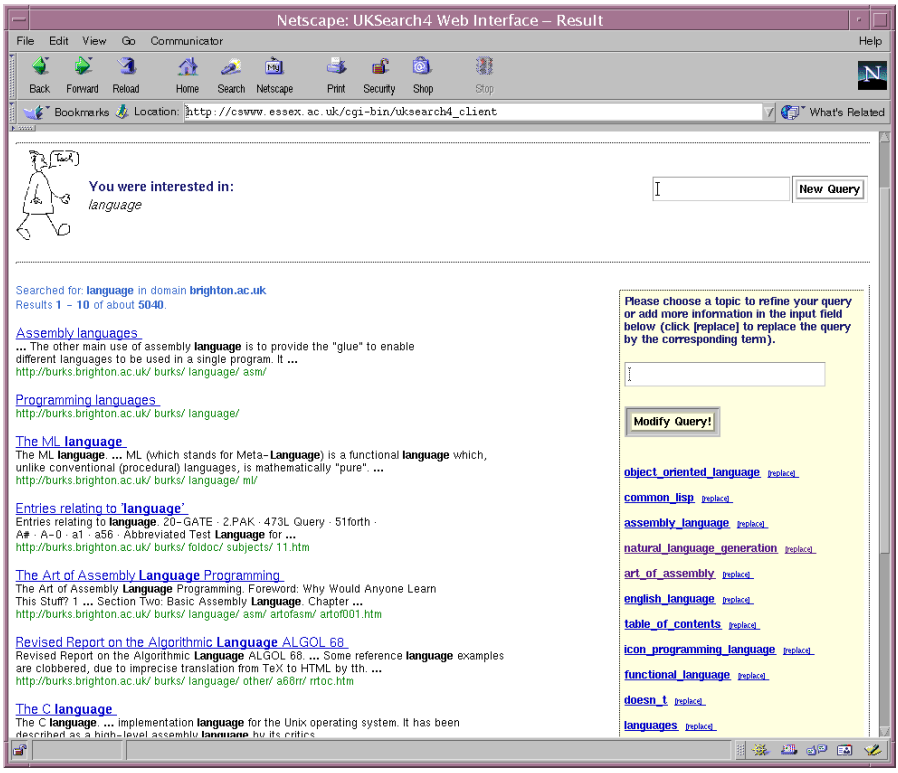

Fig. 6.11. *UKSearch*: system's response to the user query *"language"* (Brighton)

7

UKSearch - Evaluation and Discussion

How do we evaluate a hybrid system such as *UKSearch*? Clearly, simply assessing precision and recall would be misleading. This would partly be an evaluation of the search engine we incorporate. What needs to be investigated is the quality of the query modification options that are calculated based on the concepts encoded in the domain model. We have performed a number of experiments ranging from a log file analysis to task-based evaluations. We will give a detailed account of these experiments and discuss the results. There are four experiments in total which we want to discuss here:

- We shall first look at a technical investigation, namely an analysis of the Essex log files of real user queries using the existing locally installed search engine (Sect. 7.1).
- We will then present results of a first user-oriented evaluation step, that involved potential users of *UKSearch*. The subjects had to judge the usefulness of term relations uncovered in the domain model construction process (Sect. 7.2).
- The third experiment is a task-based evaluation that has been performed for the Essex domain (Sect. 7.3).
- Finally, we discuss a task-based evaluation performed for the BBC News domain (Sect. 7.4).

7.1 Log Analysis

Our first step towards an evaluation aimed at a simple justification for a more advanced search system that goes beyond pure keyword search. We started by analyzing query logs collected for the existing search engine in one of our sample domains.

This section presents an evaluation step that is purely technical. We used the first sample domain (University of Essex Web site) for the technical investigations, mainly due to convenience (e.g. availability of a query corpus of real user queries, no legal or access issues etc.).

In analysing the query logs we tried to be as objective as possible. This is particularly important since we have no idea what a user who submitted a query actually had in mind. We submitted each of the selected queries to a search engine that has sufficient coverage of our domain and investigated the result set. We had to make the simplifying assumption that the search engine returns all matches that could be relevant for a particular query and nothing else.

7.1.1 System Setup

In order to get results that take state-of-the-art technology into account we used Google as the standard search engine of our choice. We did this for a number of reasons:

- The technology behind Google [20] makes it one of the best search engines publicly available. Studies have also found it to be one of the most popular search engines (see for example [68, 164]).
- Google allows to restrict the search to a particular domain (i.e. `essex.ac.uk` in this case). Therefore, there is no need to install a separate local search engine for such experiments.
- Google's database covers our sample domain sufficiently (this can be found by comparing the number of matches for general and specific queries submitted to both Google and our database).
- The existing search engine on the Essex University Web site is not very sophisticated. Applying it in this investigation would have been less useful.

As a basis for testing we used the log files that record any query being submitted to the existing search engine at the University of Essex between 1^{st} January and 31^{st} March 2001. In total 26,365 user queries were submitted in that period.

A number of normalization and filtering steps had to be performed to select our test sets of queries. First of all, all queries were turned into lower case letters, and typographical characters were replaced by space. Secondly, all queries containing numbers (as well as empty queries) were ignored.

Two test sets of queries were chosen to conduct the log analysis:

1. The top 100 most frequently submitted queries in the evaluation period (refered to as *Set 1*)
2. All queries submitted on an arbitrarily chosen day of the evaluation period with a total of 206 unique queries (*Set 2*)

We will give some examples of how the test sets are composed. First of all, the ten most frequent queries of *Set 1* are as follows (query with corresponding frequency):

```
409 yahoo
304 summer school
226 fees
210 enlightenment
181 cmr
164 term dates
164 application form
153 timetable
147 prospectus
135 exam timetable
```

It might seem somehow surprising that the most frequently submitted query is one unrelated to the domain: *"yahoo"*. In this respect the log file shows some resemblance with standard search engines. For example, *"yahoo"* was found to be the second most frequent user query submitted through `msn.de` in the year 2000 [149]. Most other queries in *Set 1* are very domain specific.

In addition to the most frequent queries, we give the (alphabetically) first ten queries of *Set 2*:

```
accomadation
accomodation
accomodation office
acommodation office
afm
alison booth
alta vista
application
application forms
application form
```

As the example log shows, there are a number of misspelled queries. The most prominent one is *"accommodation"* in its various forms. The misspelled form *"accomodation"* can be found more frequently than the correct spelling. *Set 2* does not contain the correct spelling at all. This is a problem typical to the Web, i.e. it goes beyond the queries and equally concerns the content of Web pages. Some tests presented in [114] indicate that one in three foreign names is misspelled on the Web and that even for common words like *against* the error rate is about 0.5%.

We treat misspelled queries like any other query. After all, the user request for *"accomodation"* still retrieves more than fifty matching documents! There are however fewer spelling errors in our test set than we expected. To get a clearer picture we include a figure for that in Table 7.1 which gives some statistics for our test data.

It is interesting to note that the average length of a query submitted to the Essex University search engine is shorter than what was presented in [138] and [73] (an average query length of 2.35 and 2.21, respectively). These results were based on queries submitted to standard search engines (*AltaVista* and *Excite*, respectively). Taking the entire set of 26,365 queries submitted

Table 7.1. Test data

	Set 1	**Set 2**
Number of Queries	100	206
Average Query Length	1.45	1.78
Length of Longest Query	4	6
Queries with Spelling Errors	2.0%	3.9%
Fraction of Query Corpus	24.3%	0.8%

in the evaluation period, we get an average length of 1.72 terms per query (maximum length of 15 words). Furthermore, the top 100 most frequently submitted queries account for 6,404 of all queries submitted in the evaluation period, which is about a quarter of the entire query corpus.

7.1.2 Results

The most striking fact is the number of documents retrieved when submitting the requests to Google. In Table 7.2 we give a breakdown of what percentage of queries in each of the test sets resulted in how many matching documents.

Table 7.2. Number of documents returned by Google

	Set 1	**Set 2**
No matches	0.0%	2.4%
Between 1 and 50	16.0%	36.9%
Between 51 and 100	11.0%	8.2%
Between 101 and 500	38.0%	24.8%
Between 501 and 1000	14.0%	14.1%
More than 1000	21.0%	13.6%
Average number of matches	620	226

The average number of matches for queries in both test sets is much larger than we expected in a fairly small document collection like this one. Although the deviation from the average is very large, it can also be observed that more than one third of all frequent queries resulted in more than 500 matches, and there are only 16% of all frequent queries for which Google would return no more than fifty matches. Given the fact that a screen normally displays ten matches only and that a user hardly ever goes to the next screen at all [138], these figures are a clear argument for more advanced search engines that go beyond displaying a large number of matches.

For the queries in *Set 2* the figures are different. But more than 50% of all the queries still retrieved more than a hundred documents.

7.1.3 Discussion

To summarize, an interesting fact established by this technical investigation is that even in a fairly small domain like the one we investigated the data sparsity problem seems to be less significant than the problem of "too many" matches. Queries submitted in our sample domain usually result in a large set of matching documents despite the relatively small size of our domain. Part of the reason is that users normally submit queries consisting of one or two words only.

The results obtained from the user corpus are a justification for a query refinement approach that helps narrow down the search. However, it does not answer questions about the usefulness of the particular methods. This will now be discussed.

7.2 Investigating Domain Model Relations

A user survey has been conducted to find out whether the relations between term pairs in the domain model(s) are indeed sensible relations. In other words, would potential users of *UKSearch* find them relevant when they search the sample domain(s)?

7.2.1 Task and Setup

For this user study we adopted an approach used by Sanderson and Croft [134], who investigated how "interesting" users would find pairs of words automatically extracted in a process that constructs term hierarchies. Our study is simpler than that in that we were not interested in finding out what sort of relation exists between two terms, but whether such a relation could be "relevant" in an ad hoc search system or not. Although such a study only looks at one aspect of the domain models we construct, we feel this is one way of making the results more comparable than more complex investigations which would for example let users assess entire concept hierarchies.

Subjects were staff and students at Essex University. They were given two forms, one for each domain (i.e. Essex and BBC News). The introduction given to the subjects was the following (with a link to two forms that each contained pairs of related terms and some more instructions):

> *This is a little experiment. It consists of filling out two survey forms. You are the user of a new search engine which searches a specified document collection.*
> *In addition to returning the best matching documents for any given query, this search engine also returns a set of words or phrases (called terms)*

- *that give an indication of what the retrieved documents are about, and*
- *which can be added to the query in order to refine the search.*

The following pages give a list of term pairs. For each pair, imagine the first term was your original query, and that the second is one of the terms proposed by the search system, which you could use to refine the search. Please judge for each pair whether you think the second term is:

- *relevant (tick "Yes")*
- *not relevant (tick "No")*

If you do not know, then tick "Don't know".
Here, "relevant" means that you can imagine a situation where the second term is an appropriate refinement of the query given by the first term.
When considering relevance, remember the particular document collection that is being searched, as specified on the form.

Subjects were not told that two different techniques have been used to generate these term pairs.

Each of the two survey forms contained 50 term pairs in random order, 25 of them pairs found in the automatically constructed domain model, the other 25 pairs were pairs selected using a baseline approach. The selection process for a pair of terms was as follows:

- Using the Essex log files again, we selected the most frequently submitted queries. For each of these queries we consulted the automatically created domain model to find the hierarchies that contained the query in the root (queries consisting of more than one query term were treated as compounds). We then selected only one link from the root node down the next level, the one with the highest weight. Each term pair was then built by taking the query and the alphabetically first concept in the selected node. If there was no appropriate domain model hierarchy in the model, then we ignored the query.
  For each query selected as mentioned we also constructed a random pair as outlined further down.
- For the BBC News domain we did not have log files at the time we performed this evaluation. Nevertheless, we discussed that frequent query terms are likely to turn up as concepts in the document collection. At least for the Essex domain we found a strong overlap between concepts and queries frequently submitted to the existing locally installed search engine. Therefore we decided to make the same assumption for the BBC News domain to approximate real queries. We decided to select the most frequent type-3 concepts found in the BBC News domain. For the construction of related pairs and random pairs we followed exactly the approach used in the Essex domain.

- As a baseline we used Google's API to submit a query in the specified domain, selected the first page of matches Google returned (i.e. the ten highest ranked documents), downloaded these documents and selected the most frequent term found. For selecting a term we used the same indexing steps and the same patterns as we did for building a domain model. We also applied the same stopword list. That means we deliberately used a good baseline ignoring all terms that were not nouns or noun phrases or that were found in the list of stopwords.

Naturally, we only get a very limited picture of the usefulness of term relations. We do not even investigate the levels further down in the domain model. The pairs we have selected seem sensible because for a given query we ask the user to judge the refinement term that the system assumes to be the *best* one for this query.

Interestingly, it only happened once that for a given concept we selected exactly the same term using the domain model and the baseline approach. This was the case for the query term *ssh* (which stands for "secure shell") in the Essex domain. The most frequent term found in the result set (baseline approach) happened to be *password*. The domain model delivers exactly the same term, because the hierarchy that contains *ssh* in the root node suggests *password* as the most highly ranked query refinement option.

7.2.2 Results

In total 31 subjects were recruited for this experiment, 19 of them members of staff in the Department of Computer Science, 12 of them students from various departments.

The results confirm that the concept-based approach is a significant improvement over the baseline. In both domains the users judged more domain model based terms to be sensible refinement options than terms calculated on the fly using the baseline approach. This is how users judged the term pairs for their relevance:

- BBC News domain: 64% of the potential query refinements using the concept-based approach were considered *relevant* (for the baseline approach 48% of the potential query refinements were considered *relevant*).
- Essex domain: 59% of the potential query refinements using the concept-based approach were considered *relevant* (baseline: 50%).

There is some interesting resemblance with the figures reported in [134]: 67% judged the term pairs in the concept hierarchies to be *interesting*. The baseline approach gave 51%. In fact, the results we get are also remarkably similar to the results of a study reported in [85], where anchor text was used to construct query refinement terms, and users had to judge how "useful" these terms would be to explore, learn about, or refine the original topic. The study (called *User Study II*) which is comparable to our experiment concludes

that 64% of the query refinements that were derived from anchor text were considered useful (similar to the percentage of terms derived from query logs), a good baseline gave 51%.

In our study, the differences in judging the term pairs constructed using the two techniques were found to be significant by paired t-tests. For the BBC News data we found a significance with $p < 0.0005$ and for the Essex data $p < 0.003$. It is interesting to note that significance can also be shown when we investigate the results obtained for either students or staff members only.

7.2.3 Discussion

Although significant differences between the two approaches exist in both domains, it is obvious that the gap between baseline and concept-based approach is much wider in the BBC News domain. One explanation is that there is more consistent markup used in that domain (which allows the extraction of a "better" domain model). The other aspect is that rather than downloading the first ten matches Google returns one could download say the first 200 pages and extract potential query refinements. Nevertheless, the similarity to the figures presented in [134, 85] would suggest that the overall picture would not change much.

For the Essex domain we noticed that some users had problems with the proposed terms (sample user comment: *"Not sure what a lot of the initials stood for."*). Due to the smaller size and more inconsistent markup the domain model contains some very domain specific relationships (such as term pair *afm* and *abdala*) which users were likely to find irritating and subsequently judge them as not relevant.

Users were also invited to give additional comments. The idea was to treat them as feedback on both the user survey as well as the techniques in general. One comment that several users had was that "relevance" could not always be judged by answering *yes* or *no*, but that it should rather be judged on a scale. This was a comment that had already been identified in sample trials. However, for this experiment we wanted to obtain results that could be compared to figures reported in [134].

It should also be pointed out here that although we compared the results of the user study with Sanderson & Croft's results, these two approaches are significantly different. One is based on constructing relations *offline* for the entire document collection, the other one builds hierarchies for a small set of documents *online* as a result of a user query. Therefore, it is hoped that both approaches could benefit from each other by combining them in one search system. In fact, a number of different approaches could perhaps be combined since they might be complementary. After all Kraft & Zien established in their study that "the coverage of anchor text and query logs differs substantially enough that both are useful for query refinement" [85]. This research area will be left as an issue to be explored in the future.

7.3 Task-Based Evaluation: Essex University

Evaluating search systems is an important research issue. A problem that arises when evaluating such systems is the interpretation of the results. Results that cannot be re-validated or compared against alternative approaches are not very useful. This is why we adopted an existing evaluation framework to evaluate the complete search system. The central idea was to compare *UKSearch* with a standard search engine. The TREC conference series has been successful at setting the standards for comparing systems against each other. Since we were interested in comparing two different search systems that work on the same domain, we adopted the evaluation methods developed for the TREC interactive track [156]. We compared the actual *UKSearch* system and a baseline system. Both systems access the same document collection, the University of Essex Web site. The two systems can be characterized as follows:

- *System A* is the baseline system which functions like a standard search engine: the query is submitted by the user, Google (accessed via the Google API) returns the results and the first ten matches are displayed. A user can then either modify the query, go back to start a new search or click through the result set via a link that takes the user to the next 10 matches.
- *System B* is the *UKSearch* system that uses the automatically constructed domain model to assist a user in the search process by offering ways to relax or constrain the query. It also uses Google's API to display the best matching documents alongside the options that have been found.

The systems look almost identical; the difference is only that in *System A* no query modification options are constructed. Apart from that the GUI and functionality is exactly the same in both systems. Figure 7.1 is a screenshot of the baseline system showing the system's response to the user query *"phd"*. Figure 7.2 shows the response of the actual *UKSearch* system (i.e. *System B*) following the same query. The screenshot demonstrates that for this evaluation we did not offer the user the option to *replace* the query by a suggested modification term. However, the evaluation discussed in the next section uses a different setup.

16 subjects and 8 search tasks are needed to perform an evaluation according to the guidelines originally developed for the TREC-9 interactive track [67]. We will first discuss the experimental setup in more detail. After that we will discuss the results.

7.3.1 Search Tasks

Apart from adopting the TREC interactive track guidelines we also wanted to address one particular limitation of the experimental setup in that track, namely the fact that for the experiments the "conditions are artificial" [66].

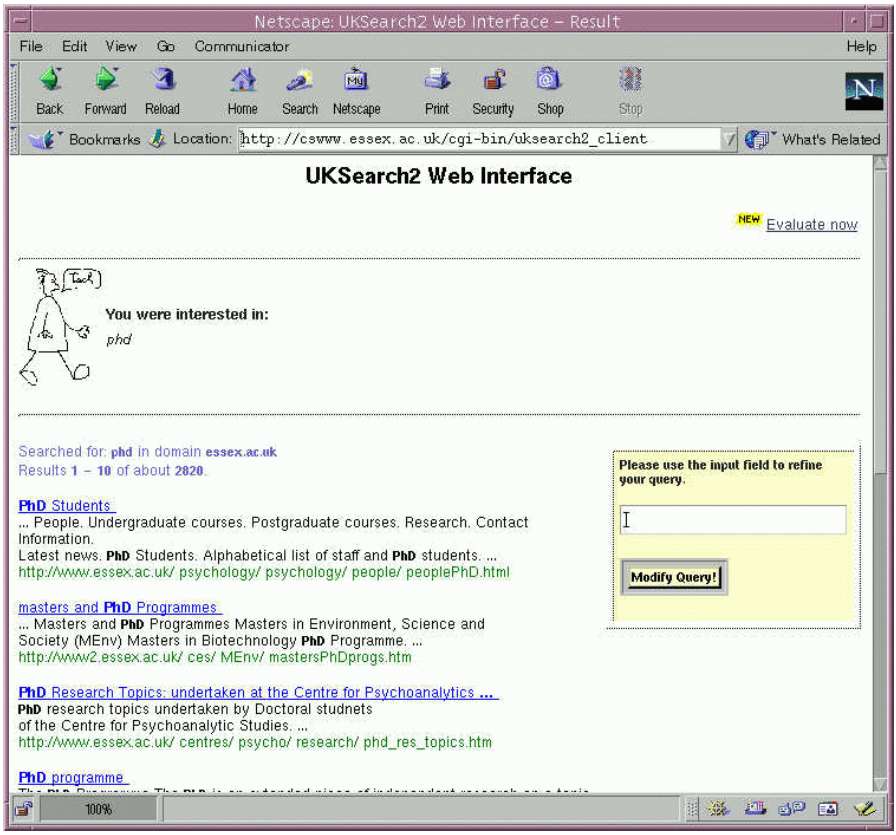

Fig. 7.1. System A: a user has typed in *"phd"*

Unlike in the TREC interactive track we did not want to construct hypothetical search tasks but use the log files of queries submitted to the existing search engine at the University of Essex to make the search tasks as realistic as possible.

Therefore, we constructed search tasks which:

- are based on *frequently submitted queries*,
- seem realistic (although we cannot be sure about the actual information need a user had by just looking at logged user queries, we can try to construct tasks that could have resulted in the query the task was derived from),
- aim at single documents that contain the answers for the particular tasks,
- are not trivial, in the sense that we tried to prevent queries that return an appropriate answer for the search task among the top 10 ranked documents

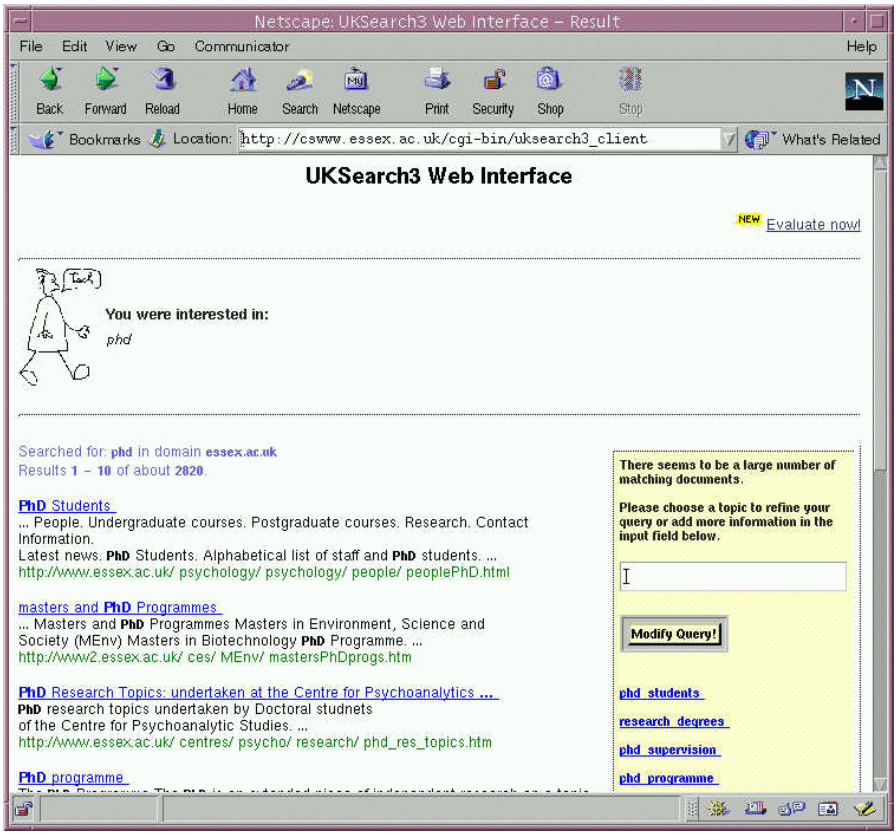

Fig. 7.2. System B: a user has typed in *"phd"*

(because in that case we would have a very short interaction with the system).

The last point in our list of requirements proved very difficult due to the fact that we used Google as the backend search engine, and the results returned by Google tend to be ordered in a way that the most sensible matches for a user query can be found early on in the list of matching documents.

To summarize, we designed tasks with fairly precise targets for which we knew documents existed that satisfy the information request (possibly more than one), and where the main difficulty would be in finding such a document rather than assessing lots of very similar documents or collecting information from different documents.

We also made sure that the search tasks were not tailored to fit our domain model. In other words, if we have access to the domain model when constructing the search tasks, then there could be a temptation to select cer-

tain tasks simply because we assume that the domain model is a useful aid in guiding the user in the search. Therefore we first constructed the tasks and then indexed the document collection and built the domain model.

The following tasks were constructed for this evaluation (the query that the task is based on is shown in brackets and is not part of the actual search task).

- **Search Task 1** (*accommodation*): Your department invites a seminar speaker from Edinburgh. The speaker will need accommodation for one night. Locate a document which contains information about suitable accommodation.
- **Search Task 2** (*password*): You have forgotten the password for your Essex account. Find a document that tells you who to contact.
- **Search Task 3** (*scholarships*): You want to find out what external scholarships are available for postgraduate students from Asia - that is scholarships *other than* those offered by the University or the individual departments. Locate a document that has information about such scholarships for study in the UK.
- **Search Task 4** (*football*): You want to play football using one of the University football pitches. Find a document which tells you what you need to do to book the football pitch.
- **Search Task 5** (three frequent queries: *vacancies*, *jobshop* and *jobs*): You are looking for a job as a student to work at the University. Locate a document which has a list of jobs currently available. Documents that only list jobs outside the University or jobs not available to students are not relevant.
- **Search Task 6** (*gallery*): Essex University has its own art gallery. Find a document which has information about what is currently being shown at the gallery (or has been shown earlier this year).
- **Search Task 7** (*phd*): The University offers a number of different research degree schemes. Imagine you are interested in doing a PhD at Essex University. Locate a document that informs you about what area you can do your PhD in and how long it typically takes to do a PhD here. Documents that inform you about PhDs in individual departments only are not relevant.
- **Search Task 8** (*student support*): Find a document with contact details about help for students who have problems with their landlord and need support. Documents that indicate help for University accommodation only are not relevant.

7.3.2 Experimental Setup

The questionnaires used in this study are based on the ones proposed by the TREC-9 interactive track (using a 5-point Likert scale where appropriate)[1]. We used the following questionnaires:

- Entry questionnaire
- Post-search questionnaire
- Post-system questionnaire
- Exit questionnaire.

Some of the questionnaires were customized to reflect the particular experimental setup. For example, in the post-search questionnaire for *System B* the user was asked whether the query modification options presented by the system were sensible.

The assignment of subjects to tasks was based on the searcher-by-question matrix displayed in Table 7.3. This table contains the mapping of tasks to searchers as proposed in [67]. Note that in the table *System A* is the baseline system and *System B* is the actual *UKSearch* system.

Table 7.3. Searcher-by-question matrix

Searcher	System: Questions	System: Questions
1	B: 4-7-5-8	A: 1-3-2-6
2	A: 3-5-7-1	B: 8-4-6-2
3	A: 1-3-4-6	B: 2-8-7-5
4	A: 5-2-6-3	B: 4-7-1-8
5	B: 7-6-2-4	A: 3-5-8-1
6	B: 8-4-3-2	A: 6-1-5-7
7	A: 6-1-8-7	B: 5-2-4-3
8	B: 2-8-1-5	A: 7-6-3-4
9	A: 4-7-5-8	B: 1-3-2-6
10	B: 3-5-7-1	A: 8-4-6-2
11	B: 1-3-4-6	A: 2-8-7-5
12	B: 5-2-6-3	A: 4-7-1-8
13	A: 7-6-2-4	B: 3-5-8-1
14	A: 8-4-3-2	B: 6-1-5-7
15	B: 6-1-8-7	A: 5-2-4-3
16	A: 2-8-1-5	B: 7-6-3-4

[1] http://www-nlpir.nist.gov/projects/t9i/qforms.html

7.3.3 Procedure

The procedure that every subject had to go through has been adopted from [37]:

- Subjects started by filling in the entry questionnaire.
- This was followed by a demonstration of the two systems. The users were informed that they will be using two different search engines, but they were not told anything about the technology behind them, nor about the use of Google as the backend search engine. Subjects were free to ask any questions. The example presented on both systems was the initial query *"union"* followed by a number of query modification steps.
- Users were then asked to perform four search tasks on one system followed by four tasks on the other one (according to the matrix in Table 7.3). Users were asked to use the (online) study worksheet for every search task to submit the result and their confidence level.
- After each task users were asked to fill in the post-search questionnaire.
- After completing all four search tasks on one system users were asked to fill in the post-system questionnaire.
- Finally, after finishing all search tasks users had to fill in the exit questionnaire.

Subjects had 10 minutes for each task. After that time they were informed that the 10 minutes had passed. Why did we restrict the search to 10 minutes? It has been found that in the TREC interactive track "there was a strong desire to reduce the time per search (previously: 20 minutes)" [67], so that the guidelines now just state that users should be given at least 10 minutes on each task [66].

7.3.4 Results

This section gives an overview of the most interesting results. In all cases, t-tests have been used for significance testing. A discussion of the results can be found in the next section.

Subjects

We had 16 volunteers for this experiment, 8 of them male and 8 female. Their ages ranged overall from 22 to 53, but apart from two subjects they were between 22 and 29 years old. The majority of the test persons (14) were postgraduate students (Master or PhD students) with various backgrounds, the others were undergraduate students. All subjects were students at Essex University when the experiments were conducted.

The average time subjects had been doing online searching was 5.9 years (most of them between 4 and 8 years). When asked for their searching behaviour, the average value was 4.75, where 5 means daily, 4 means weekly.

No one selected any other value. An interesting observation is that only one subject strongly agreed (i.e. value of 5) with the statement *"I enjoy carrying out information searches."*, two others selected 3, everybody else selected 4 (i.e. average of 3.94).

Some other interesting statistics are summarized in Table 7.4 (based on a 5-point Likert scale, where 1 means *"none"* and 5 means *"a great deal"*).

Table 7.4. Subject experience with computers and search systems

Experience	**Mean**
Using a point-and-click interface	4.38
Searching library catalogues	4.00
Searching CD ROM systems	2.75
Searching commercial online systems	2.56
Searching the Web	4.88

Search Statistics

The average length of the initial user query was 2.35 words (*System A*: 2.41, *System B*: 2.30). This figure is in fact identical to what was calculated as the average length of Web search queries [138], a study refered to earlier.

Table 7.5 gives a picture of the average completion time broken down for each task. We decided to measure the time between presenting the search task to the users and the submission of the result. In eight cases we had to ask the user to stop because the 10 minutes had passed - three times on *System A* and five times on *System B*.

Table 7.5. Average completion time (in seconds)

System	**Task 1**	**Task 2**	**Task 3**	**Task 4**	**Task 5**	**Task 6**	**Task 7**	**Task 8**
A	259.9	261.6	293.9	288.0	150.5	177.2	427.4	361.2
B	231.8	179.4	426.5	385.6	245.9	211.6	338.8	329.9

Overall, the average time spent on a search task on *System A* was 277.5 seconds, on *System B* 293.7 seconds (no significant difference has been found).

The figures show that on average users were able to find answers quicker with *UKSearch* for half of the search tasks, whereas the standard search system was quicker for the other half. There is a significance only in one case, search task 3 ($p < 0.05$). However, the search time is only one aspect and the results do not correlate exactly with user satisfaction as we will see shortly.

In Table 7.6 we break down the average completion of a task into a number of "turns", i.e. the steps it takes to find the answer for a search task. A turn can be inputting a query (or modifying the query), selecting a modification option, following the link to the next ten matches or following a hyperlink to open a document.

Table 7.6. Average number of turns to complete a task

System	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8
A	9.6	10.4	8.8	10.4	7.0	7.2	17.2	12.2
B	8.6	5.8	14.2	14.6	10.1	8.4	14.2	12.4

The figures in Tables 7.6 and 7.5 indicate that if a user needed more time on one system to complete a task, then the user would go through more turns than a user on the other system (an exception is the last task). There is only one significant result. For task 2 users needed significantly fewer turns on *System B* ($p < 0.05$).

In the post-search questionnaire users were asked to state whether they were able to successfully complete their search task. For *System A* there was only one case where this was answered with *No*, for *System B* there were three cases altogether. The inspection of the submitted document identifiers revealed that apart from these cases there were 20 documents that did not answer the search task (10 for *System A* and another 10 for *System B*) as well as three documents that answered the search task only partially (two for *System A* and one for *System B*).

Post-Search Questionnaires

After finishing each search task a post-search questionnaire had to be filled in. The questions that were common for both systems were the following (using a 5-point Likert scale, where 1 means *"not at all"* and 5 means *"extremely"*):

- *"Are you familiar with the search topic?"*
- *"Was it easy to get started on this search?"*
- *"Was it easy to do the search on this topic?"*

- *"Are you satisfied with your search results?"*
- *"Did you have enough time to do an effective search?"*
- *"Did your previous knowledge help you with your search?"*
- *"Have you learned anything new about the topic during your search?"*

Table 7.7 gives a breakdown of the results for the question *"Are you satisfied with your search results?"* For search task 8 users were significantly more satisfied with *System B* ($p < 0.01$). The other differences are not significant. The results indicate the tendency that the more difficult questions are better handled by *System B*, whereas the basic system is better at dealing with the more straightforward questions.

Table 7.7. Post-search questionnaire (user satisfaction for each task)

System	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8
A	4.12	4.50	3.38	3.88	4.88	4.50	3.00	3.00
B	3.88	4.50	4.12	3.00	4.50	4.25	4.00	4.12

Overall, users were slightly more satisfied with *System B* than with *System A*.

Table 7.8 gives figures for the other questions of the post-search questionnaires. The values are not presented for every task, but they are the overall average values. Interestingly, there is only one significant difference ($p < 0.02$) between the two systems and that is the question whether any previous knowledge of the topic has helped in the search. This could indicate, that *System B* is better suited for allowing users to apply their knowledge in the search process (note that the familiarity with the topic was very similar in both systems).

The post-search questionnaire for *System B* contained two additional questions, that were not relevant for the basic search engine. One question was *"Did you know at each point in the interaction with the system what options you had to continue the search task?"* (a type of question adopted from the *PARADISE* framework for the evaluation of spoken dialogue systems [160]). The mean value for that question was 3.66. The other question was *"Were the query modification options presented by the system sensible?"*. Here we had a mean value of 3.44. Both these results suggest that *UKSearch* is easy to use and typically presents sensible query modification terms.

Table 7.8. Post-search questionnaire

System	Familiarity	Start	Search	Satisfied	Time	Knowledge	Learned
A	2.84	3.48	3.33	3.95	4.23	2.91	3.27
B	2.98	3.70	3.56	4.05	4.30	3.47	3.21

Post-System Questionnaires

After performing four search tasks on one system a post-system questionnaire had to be filled in. Here we only present the statistics for the multiple choice questions. Later we will discuss any additional comments made by the subjects in more detail. Table 7.9 presents a breakdown of the results. No significant differences were found.

Table 7.9. Post-system questionnaire

Question	System A	System B
How easy was it to *learn to use* this information system?	4.19	4.44
How easy was it to *use* this information system?	3.88	3.88
How well did you *understand how to use* the information system?	4.19	4.06

Exit Questionnaire

In the exit questionnaire users were asked to answer the question *"Which of the two systems did you like the best overall?"*. 9 users prefered *System B*, 6 prefered *System A* and 1 found no difference. Remember, that the underlying search engine in both systems was Google, which means that *System B* essentially competed with a version of Google. Note however that the users were not told anything about the underlying search engine.

We should also stress that such a preference for the system that we propose cannot just be taken for granted. In a recent study which also compared

a baseline search system with a system that incorporates automatically constructed concept hierarchies it was found that more than half the subjects prefered the baseline system "because of its simplicity and familiarity" [75]. Perhaps the fact that we do not present the full hierarchies but simply some query modification terms alongside the search results makes our system more familiar and hence more preferable by the users.

Table 7.10 summarizes how users judged the systems in respect to learning to use and using them. The table also contains the overall preferences. Displayed are the numbers of users who selected each of the choices.

Table 7.10. Exit questionnaire (system preference)

Criterion	**System A**	**System B**	**No difference**
Easier to learn to use	6	5	5
Easier to use	6	6	4
Best overall	6	9	1

Table 7.11 summarizes the answers users gave in the exit questionnaire concerning the search experience they had in the experiment (where 1 means *"not at all"* and 5 means *"completely"*).

Table 7.11. Exit questionnaire (search experience)

Question	**Mean**
To what extent did you understand the nature of the searching task?	4.31
To what extent did you find this task similar to other searching tasks that you typically perform?	3.88
How different did you find the systems from one another?	3.38

The most important result of this evaluation is the fact that the majority of users prefered the concept-based search system over the baseline. Furthermore, users were slightly more satisfied with the search results returned by *UKSearch* than the baseline.

7.3.5 Discussion

So far we have mainly been concerned with the statistical evidence. In this section we will discuss some of the other issues, such as feedback that came from the subjects as well as patterns in users' search behaviour.

Most of the questionnaires contained text fields that could be used to give detailed feedback on tasks or systems. We will not try to discuss all issues raised. Instead, we will highlight some of the more important aspects (e.g. problems that are not restricted to single users) and present some of the lessons learnt.

Many users liked the general idea of having refinement options. Some problems with these options were that they were not always good and that there should be more help so that one knows exactly what the options are. It was also noted that more time would be required to learn to use the system more efficiently. One user commented that *System B* may be better when searching a broad topic or searching huge document collections.

Users typically liked the simplicity of *System A*. One user noted *"I like system A because it more or less reflects my web searching habits and feels more natural to use."*

A major problem - and one that can easily be rectified - had to do with the interface. Users got confused with the input field that was provided for both systems (see the screenshots in Figs. 7.1 and 7.2). A common mistake was to use that field for starting a new search, although there was some text alongside the field saying that the field can be used to *modify* the current query. There is a straightforward solution. We now provide two separate input fields, one for a new search request and one for modifying the current query. Alternatively, one would use a single field only which contains the current query so that it can be edited by the user (similar to what search engines typically do).

Misspellings are another problem. The word *accommodation* was misspelled frequently (16 out of the 128 search tasks contained queries with typos, eight on each system; of those there were 11 in tasks 1 or 8, topics that had to do with accommodation). The actual problem is the fact that there is indeed a sufficiently large number of documents matching the misspelled query. As a result the query modification options proposed by *UKSearch* can actually be misleading. A similar but domain-specific problem occured when users typed *"job shop"* instead of *"jobshop"* (although we did not consider this a typo). A possible solution for both of these problems is Google's approach, i.e. to present the results as usual but also ask the user explicitly *"Did you mean: ...?"* This would be sensible for *frequently misspelled* query terms, including domain specific terms like *jobshop*. One user suggested that the system should assist a user when the query is misspelled.

Two users ignored all the options that *UKSearch* proposed altogether and instead used the system just like a standard search engine. It is actually one of the intended features of the system that users do not get too distracted by

the modification options, but treat the system as a search engine with some additional features.

There were two users who typically used the search engine to locate a good entry page (in this case the Essex University homepage) and then navigated to the required documents from there using the link structure.

Finally, it must be said that such an evaluation is a difficult task. On the one hand it is desirable to have a selection of real users (i.e. Essex students in this case), on the other hand their previous knowledge will vary dramatically. An indicator are the queries users submitted for search task 1 (*"Accommodation"* vs. *"Wivenhoe House Hotel"*) and task 4 (*"university football pitches"* vs. *"Sports Centre"*).

A final comment on the evaluation framework we used for the experiments. As an alternative to applying the TREC interactive track guidelines for a task-based evaluation we could have adopted the *PARADISE* approach, an evaluation framework developed for spoken dialogue systems [160]. In this framework the assumption is that "values for user satisfaction could be predicted on the basis of a number of simpler metrics that can be directly measured from the system logs, without the need for extensive experiments with users to assess user satisfaction".

7.4 Task-Based Evaluation: BBC News

The user studies we looked at so far focused on two questions:

- Does the domain model encode term relations which can be useful for query refinement?
- Will users prefer an integrated system (that incorporates our domain model) over a standard search engine?

For both these questions we found supporting evidence that the general approach presented in this book is indeed a sensible one. However, we only looked at one domain (the University of Essex Web site) and so far we have *not* combined the domain model with terms extracted on the fly. This is what we investigated in the second task-based evaluation discussed here.[2] The actual experimental setup of the evaluation was very similar to the earlier experiment we performed on the *University of Essex* domain. However, we did mention that the dialogue strategy in the BBC News domain is slightly different. Apart from that we did minor modifications to the system in the light of the previous experiment. We will also look at some of the results from a slightly different angle.

[2] *Portions reprinted, with permission, from U. Kruschwitz and H. Al-Bakour. Users Want More Sophisticated Search Assistants - Results of a Task-Based Evaluation. Journal of the American Society for Information Science and Technology (JASIST), to appear.* © *2004 Wiley Periodicals, Inc., A Wiley Company.*

7.4.1 Search Tasks

We have been able to acquire a substantial corpus of queries submitted in December 2003 to the search engine installed at the BBC Web site. This corpus was used to construct the search task which (similar to the tasks constructed for the Essex domain):

- are based on *frequently submitted queries* (four of them using frequent queries submitted to the BBC News world edition, the other four based on queries to the UK edition),
- seem realistic (although we cannot be sure about the actual information need a user had by just looking at logged user queries, we can try to construct tasks that could have resulted in the query the task was derived from),
- aim at single documents that contain the answers for the particular tasks (this seems realistic for searches in a news domain and furthermore makes the creation of tasks simpler),
- are not trivial, in the sense that we tried to prevent queries that return an appropriate answer for the search task among the top 10 ranked documents.

Just like in our first evaluation we found the last point to be particularly difficult to satisfy due to the fact that we used Google as the backend search engine.

The following tasks were constructed for this evaluation (the query that the task is based on is shown in brackets and is not part of the actual search task).

- **Search Task 1** (*brazil*): You are asked to find information about the current president of Brazil. In particular, locate a document that has detailed information about what he did in the 1980's and 1990's before becoming the president of Brazil in 2002. Documents which give a general profile of the country are not relevant.
- **Search Task 2** (*iraq*): Locate a document that contains short summaries of the main political figures in Iraq *before* the last Iraq war started.
- **Search Task 3** (*aids*): The end of last year saw the start of a new big campaign to fight Aids worldwide. Find a document that gives a summary of initiatives from around the world. Documents about activities in individual countries only and documents prior to 2003 are not relevant.
- **Search Task 4** (*flu*): Some scientists say that a new global outbreak of flu is inevitable. Find a document that has details of how Britain was affected by a recent outbreak of flu and how the country coped with that.
- **Search Task 5** (*lotto* and *lottery*): Find a document that has recent, detailed examples of how the money that the UK government raised by selling Lotto tickets was spent. Information which is more than one year old is not relevant.

- **Search Task 6** (*m6 toll*): There was a lot of discussion about the first privately financed motorway in Britain that opened recently. Find a document that has information about how some of the money that users of this motorway have to pay will be used to support other projects.
- **Search Task 7** (*travel*): Imagine you want to travel abroad and you are not sure what exactly you are (or are not) allowed to take with you in your hand luggage when boarding a plane. Locate a document that has details about who you can contact to get up-to-date information.
- **Search Task 8** (*euro*): Find a document that has details about the development of the Euro currency since its introduction in 1999. Documents are only relevant if they have milestones of the Euro's development covering the entire period from 1999 till at least the end of 2003.

7.4.2 Experimental Setup and Procedure

We adopted the same TREC interactive track guidelines and again distinguished between *System A* (the baseline) and *System B* (the actual *UKSearch* system). Experimental setup and procedure were identical to the first task-based evaluation (see Sects. 7.3.2 and 7.3.3).

7.4.3 Results

This section summarizes the most interesting results of this task-based evaluation. In all cases, t-tests have been used for significance testing. A detailed discussion of the results can be found in Sect. 7.4.4.

Subjects

The aim was to find a set of volunteers who could be potential users of a search engine such as *UKSearch*. In order to get a good selection of different types of users and to avoid any bias in the selection process we sent an email to the local University mailing list and selected the first 16 volunteers who replied. In the email we specified that we required the subjects to be native speakers. This is different to our first task-based evaluation.

Out of the 16 volunteers 6 were male and 10 female. Their ages ranged overall from 20 to 46 (average age 27.7). This time we had a variety of backgrounds, which included 11 students from different departments (including Literature, Electronics, American Studies, and Computer Science) as well as members of staff (clerical as well as academic). None of the subjects had taken part in previous studies in online search. The average time subjects had been doing online searching was 5.6 years (12 of them between 5 and 10 years, but there was also a user who stated 0 years). When asked for their searching behaviour, the average value was 4.75, where 5 means daily, 4 means weekly. No one selected any other value (this observation as well as the average value

are identical to our first evaluation experiment). And again, only one subject strongly agreed (i.e. value of 5) with the statement *"I enjoy carrying out information searches."*, one selected 2, four subjects selected 3, everybody else selected 4 (i.e. average of 3.69).

Some other interesting statistics are summarized in Table 7.12 (based on a 5-point Likert scale, where 1 means *"none"* and 5 means *"a great deal"*). Particularly interesting is that all users had "a great deal" of experience using a point-and-click interface and searching the Web whereas hardly anyone had experience searching on commercial online systems.

Table 7.12. Subject experience with computers and search systems

Experience	**Mean**
Using a point-and-click interface	5.00
Searching library catalogues	3.27
Searching CD ROM systems	2.81
Searching commercial online systems	1.31
Searching the Web	5.00

Search Statistics

The average length of the initial user query was 2.89 words (*System A*: 2.82, *System B*: 2.95), i.e. longer than queries typically submitted to Web search engines (2.35 words on average according to [138]). There was one query of length 8 and one of length 7, all others were shorter than that. In our first evaluation we observed shorter queries, an indication that the tasks we set this time were more specific and perhaps more difficult.

Table 7.13 gives a picture of the average completion time broken down for each task. If the search was not completed after 10 minutes, we asked the user to start with the next task. The users could then either submit a document (and possibly indicate a low confidence) or not submit anything. There were 8 cases in which users did not submit any answer (5 on *System A* and 3 on *System B*). In those cases we added a 60 second penalty to the 10 minutes as suggested by [116] in a similar evaluation task. In Table 7.13 the t-test gives us one significant difference (task 1: $p < 0.05$).

Overall, the average time spent on a search task on *System A* was 363.5 seconds, on *System B* 383.4 seconds. There is no significant difference. Note that the average time spent on a search task is much higher than in our first experiment (around 285 seconds on average) which again suggests that the tasks were more difficult.

Table 7.13. Average completion time (in seconds)

System	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8
A	230.0	501.1	416.5	289.0	291.4	424.8	319.4	435.8
B	331.1	490.1	521.6	316.9	291.5	511.4	222.5	381.8

There are a number of other aspects that can be derived from the logged data. One particularly interesting fact is that users had to inspect fewer documents on *System B* to finish a task (on average 6.9 which compares to 8.0 on *System A*). No significant differences could be found however.

A second notable aspect is the fact that with guidance by the system the user is able to complete the task in fewer steps, although the difference is marginal and the average values per task vary a lot. In Table 7.14 we break down the average completion of a task into a number of "turns", i.e. the steps it takes to find the answer for a search task. A turn can be inputting a query (or modifying the query), selecting a modification option, following the link to the next ten matches or following a hyperlink to open a document. On average users needed 12.7 turns on *System B* (compared to 13.2 on *System A*). Tables 7.15 and 7.16 present a more detailed picture and include the average number of documents inspected for each task.

Table 7.14. Average number of turns to complete a task

System	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8
A	7.9	15.1	13.0	10.2	9.1	17.4	12.1	20.5
B	10.9	17.9	20.1	9.6	9.1	12.1	7.4	14.6

We find that the user is much more likely to come up with a manual query modification or reformulation than to select a term suggested by the system. This observation is consistent with earlier studies that looked at the users' search behaviour on the *IBM Trevi* intranet search engine [85] and *AltaVista's Prisma* tool [7]. As to the query modification suggestions presented by the system, users did indeed make use of them although not excessively (this reflects one of the objectives of our search framework which is to allow users to ignore the suggestions and instead use the system as a standard search

Table 7.15. Average number of turns to complete a task on *System A*

Type of Turn	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8
Reformulations (incl. "next 10")	2.8	6.0	4.8	3.9	4.9	7.4	5.8	5.6
Documents viewed	5.1	9.1	8.2	6.4	4.2	10.0	6.4	14.9

Table 7.16. Average number of turns to complete a task on *System B*

Type of Turn	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8
Model suggestions selected	1.6	0.6	1.5	1.2	1.1	0.4	0.4	0.4
Baseline terms selected	0.0	0.5	0.9	0.0	0.5	0.1	0.1	0.4
Reformulations (incl. "next 10")	4.6	6.6	6.4	2.8	4.1	4.6	2.4	5.1
Documents viewed	4.6	10.1	11.4	5.6	3.4	7.0	4.5	8.8

engine if they want to). On average users selected 1.2 query modifications in each task performed on *System B*. These modifications include query relaxations, refinements or replacing the current query by some suggested term. The breakdown indicates that in total the modification suggestions based on the automatically constructed domain model and the dialogue manager were selected nearly three times as often as terms which were selected from the best matching documents on the fly (the "baseline terms") although there is a big variation across the tasks. More discussion of this particular aspect of the system will follow in Sect. 7.4.4.

In the post-search questionnaire users were asked to state whether they were able to successfully complete their search task. For *System A* 9 answered with *No*, for *System B* there were 5 cases altogether. However, we also went through every submitted document identifier and judged (based on the information contained in the sample documents we identified at task construction time and on the actual search task) whether we would consider it a match for the task. We found that there was a large number of submitted documents that did not *exactly* match the information request as specified by the task. That includes partial matches or documents that did not match the outlined constraints. Only 83 of the 128 search tasks resulted in exact matches (43 on *System A* and 40 on *System B*). That is a clear indication that the tasks were very difficult. There was no significant difference between the two sys-

tems in that respect, but there were two particularly difficult tasks: tasks 2 and 6 (clearly reflected by the user satisfaction values in Table 7.17 further down). Only 3 of the 16 users found a correct document for task 6, and 5 were correctly submitted for task 2. Typical comments in the post-search questionnaire for task 6 were *"Found information about the new Motorway but not how the funds are used"* and *"found relevant information but no answer to specific question of local project benefit"*.

If we look at all 45 unsuccessful tasks in detail and compare it against the results reported earlier on, we find that on average users looked at more documents (*System A*: 10.6, *System B*: 9.1) and needed more turns (*System A*: 17.6, *System B*: 15.3). Interestingly, despite longer interactions with the system users selected fewer modification options presented by *System B* (1.0 on average per task; domain model suggestions were selected exactly twice as often as terms extracted on the fly).

Post-Search Questionnaires

After finishing each search task the users filled in the post-search questionnaire (this questionnaire was discussed in more detail in Sect. 7.3.4). All questions were answered based on a 5-point Likert scale (where 1 means *"not at all"* and 5 means *"extremely"*). Table 7.17 gives a breakdown of the results for the question *"Are you satisfied with your search results?"* Overall users were marginally more satisfied with the results returned by *System A* than with *System B*, but no statistical significance can be observed for the overall result or in fact any of the data reported in Table 7.17.

Table 7.17. Post-search questionnaire (user satisfaction for each task)

System	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8
A	4.38	2.88	3.12	3.38	3.25	2.62	3.38	3.38
B	4.00	2.38	2.88	3.25	3.25	2.38	4.38	2.75

Table 7.18 presents the results for the question *"Have you learned anything new about the topic during your search?"* Overall users indicated that they have learned more when using *System B* than using *System A*. Significant is that users learned more when performing task 4 on *System B* than on the baseline system ($p <= 0.01$).

Table 7.19 gives figures for the other questions of the post-search questionnaires. None of the differences are significant. It is interesting to point out that every single value in Table 7.19 is smaller than the corresponding value

Table 7.18. Post-search questionnaire (something learned)

System	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8
A	3.25	2.25	2.75	2.00	2.75	2.50	2.12	2.50
B	3.00	2.25	3.00	3.38	2.75	3.12	2.75	2.38

calculated in our first task-based evaluation. This is the strongest evidence that the tasks we asked users to perform in this evaluation were difficult indeed. That sentiment is also reflected by a number of comments the users provided in the questionnaires, e.g. *"It was hard to search for something that I don't know very much about"*, *"Found it hard to find the specific page."*, and *"I think most of the documents requested are specific, so it's not that possible to just search and obtain them."*

Table 7.19. Post-search questionnaire

System	Familiarity	Start	Search	Satisfied	Time	Knowledge	Learned
A	2.47	3.41	3.22	3.30	3.77	2.27	2.52
B	2.27	3.23	3.08	3.16	3.43	2.30	2.83

Now it could be desirable to quantify the complexity or difficulty of a task, but constructing tasks of different complexity is an art on its own. We do however know that task complexity can have a significant impact on issues such as search success and user satisfaction [15]. We will not try to establish different levels of complexity in our tasks now that the evaluation has been performed. However, in Table 7.20 we give a task-by-task breakdown of some of the properties displayed in Table 7.19. Here we do not distinguish between the two systems, because we want to get a picture of what the users' perceptions were about the difficulty of the tasks in general.

There is a clear pattern which shows that the higher the value for user satisfaction the easier users found it to get started *and* to do the search. The only task that does not strictly follow this pattern is the task with the lowest average user satisfaction value (i.e. task 6). But note also that there seems to be no obvious correlation between familiarity with a topic and the

Table 7.20. Post-search questions (task-by-task)

Criterion	Task 1	Task 2	Task 3	Task 4	Task 5	Task 6	Task 7	Task 8
Familiarity	1.38	2.50	2.50	2.44	2.56	1.62	3.19	2.75
Start	3.88	2.62	2.94	3.62	3.50	2.88	3.88	3.25
Search	3.88	2.38	2.62	3.44	3.44	2.88	3.69	2.88
Satisfied	4.19	2.62	3.00	3.31	3.25	2.50	3.88	3.06

"difficulty" of a task. For example, task 1 is the one users were least familiar with but it is the task they judged to be easiest to get started with and do the search. One would perhaps expect a close correspondence between these features (i.e. easier to search if more familiar) as for example reported in another experiment performed as part of the TREC interactive track: "It appears that an inherent feature of a difficult topic is the level of familiarity and the understanding of the content and context of the issues." [13].

The post-search questionnaire for *System B* asked the user (using a 5-point Likert scale) *"Did you know at each point in the interaction with the system what options you had to continue the search task?"*. The average value was 3.77. For the second question specific to *System B* (*"Were the query modification options presented by the system sensible?"*) we had an average value of 3.16. Again we conclude that users typically found the system easy to use and judged the query modification suggestions to be sensible. Having said that we also observed that although users may find the modification options sensible, they have not been used heavily. A clue could be that the tasks were rather specific (e.g. sample user comments: *"Found a great deal of general information but difficult to find exactly what the task required, even with the search options"*, and *"Some of the links given for the second system weren't that relevant to the current search"*). Another explanation is that the suggestions are useful not just for choosing the suggested query modifications but to get a feel for the document collection, e.g. one user commented *"With the extra option list offered by the system, I found it far easier to use the suggestions offered by the system or using the ideas and word combinations it gave me to modify my search."*

Post-System Questionnaires

Table 7.21 gives a breakdown of the results obtained from the post-system questionnaires. The only significant result is that users found *System B* easier to use than *System A* ($p < 0.006$). This is an important result since we consider the baseline system (*System A*) to be very simple and easy to use like most standard Web search engines.

Table 7.21. Post-system questionnaire

Question	System A	System B
How easy was it to *learn to use* this information system?	4.31	4.19
How easy was it to *use* this information system?	3.53	4.38
How well did you *understand how to use* the information system?	4.19	4.12

Exit Questionnaire

According to the exit questionnaire users strongly prefered *System B*. Despite the fact that users were slightly more satisfied with the results returned by the baseline system, their overall perception was in preference of *System B*. 13 users prefered *System B*, 1 prefered *System A* and 2 found no difference. Although our first evaluation indicated the same preference we did get a much more significant difference this time.

Furthermore, a large majority of users also judged that *System B* was easier to use than the baseline system. Finally, when looking at the question of which system was easier to learn to use it should be remembered that there is a set of different options which *System B* could come back with. Apart from presenting query refinements it may instead present query relaxations. Getting used to these options involves a short learning process that the baseline system does not require. Therefore it might be surprising that not more people voted for *System A* to be simpler to learn to use.

Table 7.22 summarizes these results. Displayed are the numbers of users who selected each of the choices. The figures in this table confirm the results of the post-system questionnaires in that users found *System B* much easier to use whereas *System A* was a tiny bit easier to learn to use.

Table 7.22. Exit questionnaire (system preference)

Criterion	System A	System B	No difference
Easier to learn to use	5	4	7
Easier to use	2	10	4
Best overall	1	13	2

Table 7.23 summarizes the answers users gave in the exit questionnaire concerning the search experience they had in the experiment (where 1 means *"not at all"* and 5 means *"completely"*).

Table 7.23. Exit questionnaire (search experience)

Question	**Mean**
To what extent did you understand the nature of the searching task?	4.25
To what extent did you find this task similar to other searching tasks that you typically perform?	3.56
How different did you find the systems from one another?	3.06

The main conclusion that we derive from the statistical evidence is that in the given context of searching for documents in a large collection of news articles users strongly prefer a search system that offers more than just a ranked list of documents. Our users favour a system that offers query refinement and relaxation options and guides them through the available information. We can further conclude that users consider such a system to be significantly easier to use. The second interesting conclusion to be drawn is that users favour such a system although there was no significant difference between the two systems according to a number of different measures.

We can also conclude that the presented query modification options were generally considered sensible although in the given setup they did not result in statistically measurable benefits. We hypothesize that the major reason for that is that the presented modification terms were not always helpful for the extremely specific tasks that had to be answered in this context.

7.4.4 Discussion

Having analyzed the statistical evidence we now want to focus on patterns in the users' search behaviour as well as feedback that came from the subjects.

Patterns in User Behaviour

First of all we wanted to know whether users make use of any suggestions presented by *System B*, be it for refinement, relaxation or replacement. We found that only a single user did not make use of any such options. Remember that users were free to ignore any options and could use the system just like a standard search engine. The fact that users actually utilize the suggestions proposed by the system confirm the observations in our first experiments.

Now if we look at it in more detail we find:

- 14 users selected *refinement* terms presented by the system. Those are terms which are added to the current query. Of those 14 users there were 9 who selected terms which have been derived from the domain model, i.e. *concepts* that have been extracted in the model construction process. As outlined earlier on we present such concepts alongside terms extracted on the fly from the best matching documents.
- 13 users selected *relaxation* options suggested by the system. Those options are presented to break down the query into individual parts for example.
- 8 users selected *replacement* options, i.e. they replaced their query by one of the refinement terms suggested by the system.

As outlined earlier, in those cases where our search system presents refinement suggestions we display a list of terms that could be added to the query, and the first in the list will be those derived from the domain model (followed by terms extracted on the fly). The intuition is based on our experiments that found terms encoded in the domain model to be more suitable as query refinement options than those extracted on the fly (Sect. 7.2). We pointed out that the domain model concepts were selected much more frequently than the terms extracted on the fly, but we do not know to what extent this might have been affected by the order of term presentation.

We also observed that the tasks seemed to be particularly difficult if we look at the figures for measures such as time taken and user satisfaction and compare them with our earlier evaluation. Obviously, we deliberately avoided simple tasks because we did not want users to type in a query and get the results back straightaway. However,when evaluating search systems on news documents one should perhaps mix simple with more difficult tasks, because we assume that simple tasks are the majority of realistic user requests, e.g. a lot of users who submit the query *"iraq"* will be interested in what is going on in Iraq right now, something that will typically be listed on the first result page.

Although the information requests we constructed were very specific, we found that with one exception all the original queries that the tasks were based on in the first place were actually submitted by at least one user (partly as the initial query, partly to replace some original query by a new one). The exception is *"travel"*, in which case we only found similar queries such as *"travel guide"* and *"travel information"*.

In our first evaluation experiments we noticed a large number of typos in the user queries. And again, nearly 20% of all search tasks in this evaluation (24 out of 128) contained queries with typographical errors, e.g. *"govenrment"*, *"wordwide"*, *"AIDS compaign"*, *"britain coped flue"*, *"sadam"*, and *"euro currancy"*. This is a significant number and indicates again the type of problems to be addressed by a search engine.

We shall now look at some of the individual tasks since a task-by-task analysis can lead to interesting differences in the interaction style [13]. We selected three tasks which according to Table 7.20 represent the most difficult

task we asked users to perform (task 2), the easiest one (task 1) and one in the middle (task 5). We are mainly interested in the interaction with *System B*.

- **Search Task 1** (*brazil*): This was the task where the average user satisfaction value was the highest and where users submitted the shortest initial queries on average (2.44 words). Only one submission was incorrect. If we only consider *System B*, we find that five users made use of query modification suggestions proposed by the system (all but one of them more than once). Interestingly, the query refinement terms that were selected came all from the domain model. The log files suggest that a typical query such as *"president brazil"* or *"President of Brazil"* was handled nicely by the system; two users selected a refinement term immediately after submitting a query (*lula_da_silva* and *luiz_inacio_lula*, respectively), both of which were followed by more selections of system suggestions later on. One user first modified the query using the input field before selecting *lula_da_silva* as a refinement term. The other two users who opted for automatically generated query modifications first rephrased their queries before choosing a relaxation option (*"Search only for: brazil and president"*, or *"Search only for: president"*, respectively). These last two cases suggest that if the query contained additional words which were not filtered as stopwords (e.g. *"Current president of brazil"*) or contained typos (*"President of Brasil"*), then no refinement option would be displayed and users needed to rephrase their query or select relaxations.
  This is also the task where a number of users typed in dates (e.g. *"1980-1990"*) which were ignored by the system which led to confusion on the users' end.
  In general users who performed this search task on *System B* considered the query modification suggestions sensible (mean value 3.75 on the 5-point Likert scale, one user selected a 1, i.e. *"not at all sensible"*, all others selected 3,4 or 5).
- **Search Task 2** (*iraq*): This was the task with the longest average initial query length (3.88 words), and at the same time one of the two tasks where the success rate was really low (only 5 correct submissions) as well as the user satisfaction value. Typically, users submitted fairly specific queries but could not find any matching documents, so that they then rephrased the query (e.g. sample queries are *"main political figures in Iraq"* and *"political figures in IRAQ pre-war"*). Five users of *System B* selected query modification suggestions, one of them only once, the others twice. Interestingly, all three correctly submitted results on *System B* were submitted by one of those five users. However, it appears that the suggestions returned by the system were not always helpful. Only in two cases did the users select a system suggestion following the initial query, in both cases they selected a relaxation which happened to be the same one each time (*"Search only for: iraq"*).

We noticed that it was difficult to find documents that deal with information for the specified time. Most of the documents seem to be about current affairs. We also assume that one of the reasons for not being able to locate a suitable document easily is because words such as *"before"* and *"pre"* would be filtered as stopwords.
Asked whether the query modification options generated by *System B* were sensible we get a very homogeneous result: average value 3.00 with one users scoring a 2, one user scoring 4, all others selected 3.

- **Search Task 5** (*lotto* and *lottery*): The average length of an initial user query was 2.62 words. Only one submission was not considered correct. Four users selected query modifications suggested by *System B*, the other four did not select any of those options. Interestingly, all those users who did choose such an option did that more than just once (up to five options). Some users of *System B* submitted queries which did not require them to do any resubmissions or modifications, but a simple inspection of the returned documents was sufficient to perform the task (e.g. *"Lotto tickets UK government"* and *"lottery funding"*). Three of the four users who selected query modifications did in fact choose a query relaxation in the first step (e.g. a user submitted *"Lottery projects 2003"* as the initial query and then selected the search option *"Search only for: lottery"*; similarly the query *"Lotto money spent"* was broken down into relaxation options and the user selected *"Search only for: lotto"*). In all three cases the user later added a term suggested by the system in a refinement step (e.g. *lottery_cash*).
For this task the user opinion as to whether the query modification suggestions are sensible varied a lot (average value 3.00, but the values ranged from 1 to 5).

The picture we get by analyzing the tasks is that in any one task about half of the users apply the suggestions proposed by the system. Furthermore, if users do select query suggestions, then they typically do it not just once for each task. It also appears that in some tasks users were more likely to relax their initial query whereas in other tasks they refined it. We do not want to over-interpret these results and instead focus on the comments that users had after using the systems.

User Feedback

Before we do so we should add that one of the criticisms in the first task-based evaluation was that users were confused by the input field in the old system which was wrongly assumed to start a new query, but in fact it was there to modify the query. This time we provided two separate input fields (one for modifying the query and one for a new query). This has been appreciated because the common feedback was that both systems were easy to use and very intuitive. In fact, a frequent comment made by the users in the exit

questionnaire was that both systems were simple to use. The simplicity of the layout was noted and the fact that there were no distractions on the screen.

Users generally liked the idea of refinement options. Several users expressed that such options can give the searcher an idea of what information is available and that the system can suggest *"similar topics that weren't originally thought of"*. Other users commented that the offered terms could either be selected or help in getting an idea of how to modify the query. One user said that *System B "helps with the spelling of keywords and can give the user more specific knowledge about the topic they are searching for."* Other comments concerning the strong points of *System B* were *"System B had refinements which really helped"*, *"This system provided more search alternatives, that yielded a wider number of relevant search results"*, and *"No clutter - just the search box and results."*

A criticism of *System B* was that the query modification options were not always helpful and that they could actually divert the search to completely different topics. One user noted: *"I liked the idea of refinement options, just didn't find them useful."* Another comment was: *"The modified query options it gave didn't always help the search - sometimes you ended up going round in circles if you weren't familiar with the search topic already. It is better than with no options however like the first system as long as you don't rely on them."*

One problem we identified was that we did not index on dates and therefore such information was simply ignored, which meant that users were not happy if they asked for *"2003"* and the input was simply deleted. That problem can easily be fixed. Another aspect specific to the domain was that users would have found it more helpful if there had been a date associated with the documents or the documents had been sorted by date.

As a confirmation of what we observed earlier a user commented *"The topics were very specific and it was quite difficult to find an exact match for any of the questions."*

So what did the three users say who prefered the baseline system or had no preference at all? The user who prefered the baseline system criticised that the options which are suggested by *System B "are not complete, therefore can tempt the user to use them, diverting the search"*. Of the two users with no preference for either of the systems one noted that *System B* was in fact more useful in breaking down the query with refinements, but that *"sometimes the refinement options were not useful"*. The other user commented that both systems were not really different to what he or she is used to and *"it is just as easy to type in your own refinements. However, when searching a subject that you are not familiar with it may help to have refinement suggestions."*

We conclude that although our users strongly prefered *UKSearch* over the standard search approach, there is still enough scope for improvements, in particular when it comes to the relations encoded in the domain model. Our vision is a domain model that adjusts automatically to the users' search

behaviour and preferences. Such a system will be outlined in the final chapter (*Future Directions and Conclusions*).

8

YPA - Searching Classified Directories

The YPA is a natural language dialogue system which guides a user through a dialogue in order to retrieve addresses from classified directories such as the *Yellow Pages*. The YPA is similar to *UKSearch* in respect to the partially structured input data (of limited size) and the dialogue driven interaction between user and system. It differs however in that the classification structure found in classified directories is an implicit part of the raw data. This system was developed at the University of Essex as a research project funded by BTexact research labs[1]. The project started in January 1997 and finished in December 1999. It is historically interesting in this context for the following reasons in particular:

- The original idea of a dialogue system to access partially structured data was the principal goal of the YPA project. This is where the idea for the topic of this book was born.
- The YPA system demonstrates that the same framework that has been used in *UKSearch* can be applied in a very different context: to search classified advertisements for which some classification structure is available in explicit format. Because such knowledge is already available, the focus is on the dialogue driven search system whereas *UKSearch* focused on the construction of a suitable domain model and its application.
- There are obvious differences between *UKSearch* and the YPA when it comes to the construction of query modification options. *UKSearch* simply matches the input against a domain model. In the YPA on the other hand we have (apart from the simple domain model) a structured database and make assumptions about the types of information stored in that database.
- Several evaluation steps have been performed for the YPA which demonstrate the usefulness of a simple dialogue system for searching partially structured document collections. Those are however not discussed here, they can be found in [161].

[1] `http://www.btexact.com`

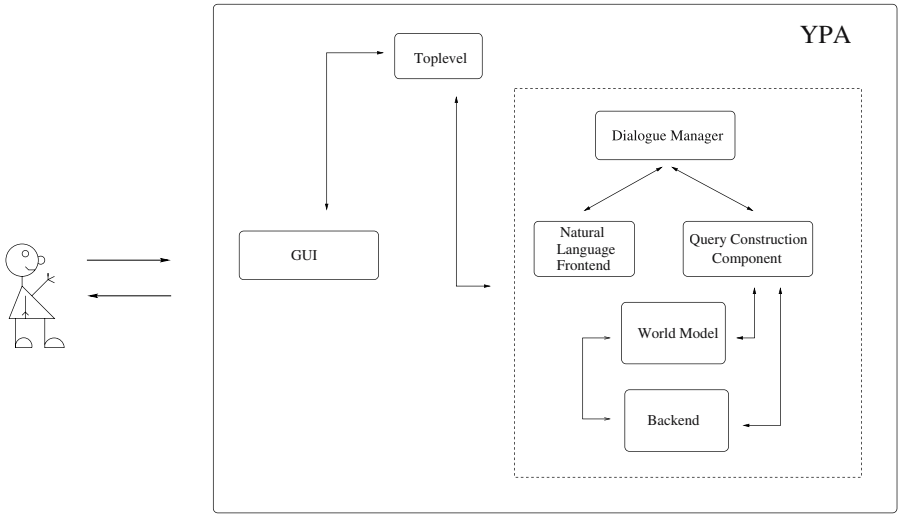

Fig. 8.1. Architecture of the YPA

We will only discuss certain aspects of the YPA here. The focus will be on how the YPA can be seen as realization of our general search framework. We will present work that has been developed and implemented by the author of this book. The YPA system is described in detail in [44].

The following transcript is a sample conversation process:

system: Welcome to the YPA. What service do you require?
user: I want to buy something in Colchester.
system: The database could not be queried.
What sort of goods do you want?
user: Some top brand golf equipment!
system: The following addresses could be retrieved:
...

8.1 System Overview

The YPA is an interactive system. Figure 8.1 is an overview of the system architecture (depicting the data flow).

A conversation cycle with the YPA can be roughly described as follows. A user input (typed in via the *Graphical User Interface*) is sent to the *Dialogue Manager*. The *Dialogue Manager* keeps track of the current stage in the dialogue and controls the use of several submodules. Before handing back control (together with the relevant data) to the *Toplevel*, the input is first sent to the *Natural Language Frontend* which returns a so-called *slot-and-filler query*. The

Dialogue Manager then consults the *Query Construction Component*, passing it the result of the parsing process (possibly modified depending on the *Dialogue History* etc). The purpose of the *Query Construction Component* is to transform the input into a *database query* (making use of the *Backend* and possibly the *World Model*), to query the *Backend* and to return the retrieved addresses (and some database information) to the *Dialogue Manager*. Finally the *Dialogue Manager* hands back control to the *Toplevel* which for example displays the retrieved addresses. It could also suggest query modifications which were passed to it by the *Dialogue Manager*, if the database access was not successful (i.e. did not result in a set of addresses). At this stage the cycle starts again.

8.2 Indexing Classified Advertisements

The electronic version of the *Yellow Pages* we had access to is represented by the so called *Yellow Pages data file* or *printing tape* in electronic form. (We were given access to the Colchester area.) All addresses, headings (i.e. business classifications) and various references are stored in a record structure, where each line is one record. However, there is no *1:1* relation between records and entries — address entries, heading entries etc. More specifically, most addresses only stretch over one line (so called *free entries*), but some addresses of a different type (e.g. *semi display entries*) will always consist of more than one line. An interesting part of these address entries is the *free text* which is an optional natural language portion to be printed in the advertisement along with the address as in the following example:

Kruschwitz Golf & Leisure Wear

Suppliers Of All Top Brand Golf Equipment

100 High Street Colchester 822990

There is no special attribute that marks a record as *free text* (in the example: *Suppliers Of All ...*). Thus, we can consider the *Yellow Pages data file* as some *partially structured* text that fits into our framework because:

- the record structure means that we do not deal with unrestricted text;
- the record structure is relatively poor, so that while the extraction of an entry is straightforward, it is not obvious how to split its address (for instance) into units such as company name, location, free text etc.

After analyzing the source file, the initial task of exploiting the given structure can be summarized in the following steps:

- layout analysis, which transforms the input file into a canonical form (similar to what we did in *UKSearch*);

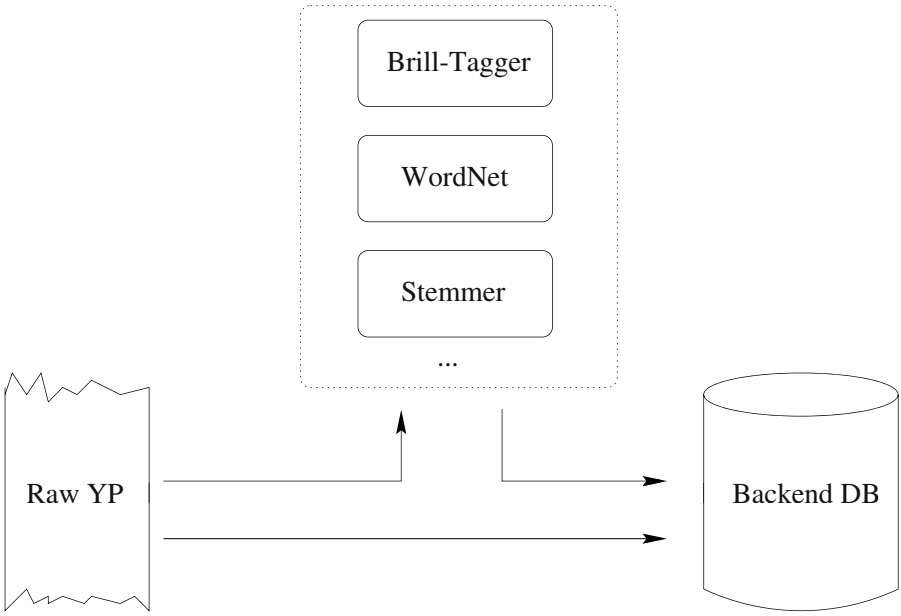

Fig. 8.2. Extraction of the YPA-Backend

- address extraction;
- extraction of headings;
- splitting address entries into smaller units (including free text);
- conversion into relational schemata.

Since we are dealing with advertisements we can apply extraction methods customized to the domain which allow us to interpret individual parts of an entry as some meaningful units (e.g. name, location, type of goods or services). The above steps will give us a more structured database than what we have seen in *UKSearch*. The result is a relational database which forms the main part of the *Backend*. Figure 8.2 reflects the data flow in the index construction process for each part of the advertisements. Note that due to the limited size of available data we do not just apply a part-of-speech tagger, but a lexicon to reduce words to their base forms and a stemmer as well. A more detailed account of the *Backend* construction processes can be found in [43]. We will briefly look at the structure of the *Backend* and then discuss how we construct a domain model (as part of the *World Model* component of the YPA).

8.2.1 Structure of the Backend

The *Backend* contains the information from the raw *Yellow Pages*. There are three subcomponents that form the *Backend*:

- the *Relational Database*, which contains all the information extracted from the raw data file (the actual addresses, indexes etc.)
- the *Ranking Component*, which contains information about the extracted data (occurrences, term frequencies etc.)
- the *Language Module*, which provides base forms and word stems for any word form.

The Relational Database

The main purpose of the *Relational Database* is to represent the addresses that were extracted from the *Yellow Pages*.

Tables exist for address entries (complete addresses, company names, keyword indexes for the free text of addresses, keyword indexes for the company names etc.) and for headings used in the *Yellow Pages* (complete headings, keyword indexes, *see*-references and *see-also*-references).

Additionally there are tables for relations which are not directly retrieved from the data file but derived by adding the information contained in the back cover of the *Yellow Pages* together with variations in the usage of town names detected in the actual addresses: *town indexes* and *dialling codes*.

The Ranking Component

The tables in the *Ranking Component* contain *meta information* about the data from the *Yellow Pages*, i.e. information about the distribution of keyword indexes in either the headings or addresses etc. This data is for example used by the *Query Construction Component* to determine the weights for retrieved addresses or the weight of potential query modifications.

Language Module

The *Language Module* provides interfaces for the morphological reduction of *word forms* to *base forms* and for the reduction of *base forms* to *word stems*. These functions are useful when trying to access the *Relational Database* or the *Ranking Component* since the indexes are constructed using the *Language Module*.

The implemented *Language Module* accesses the *WordNet* library. These functions could instead be supplied by any other system, but in this case the *Backend Databases* would have to be rebuilt in order to match the stemmed base forms stored in the index files with the base forms used in the online dialogue.

8.2.2 Domain Model Construction

We could apply exactly the same domain model construction process as we have seen in the *UKSearch* system. The different markup contexts are *classifications*, *free text*, *business names* etc. In fact, this has been done when

running the YPA system with Talking Pages data. But more importantly, in this application we can utilize some existing classification structure and do not need to construct a domain model from scratch. We can treat the headings and cross-references between them as a very reliable knowledge source. Therefore, the existing classification structure is used to construct a domain model as it was introduced in Sect. 4.1 when we discussed the incorporation of additional knowledge. We treat index terms extracted from classifications as concepts and cross-references between classifications as relations between the individual index terms. This gives us a domain model that is simply based on the relations outlined in Definitions 4.1, 4.2 and 4.3 using concepts selected from the actual classifications as we have discussed it earlier in Sect. 4.1.

Let us look at this process in some more detail. The *Yellow Pages* are divided into sections which each list a set of addresses. The name of such a section is a *heading*. Moreover, there are two types of *heading references* (i.e. cross-references) that can immediately follow such a *heading*. First, *see*-references occur in sections that do not contain addresses at all but instead only refer to other headings (heading *Fishing Agents* contains only a reference to *Shooting and Fishing Agents*). Second, there is a type of *heading reference* which is called a *see-also*-reference. This type can be found in sections which do contain addresses but where a reference might be a useful addition to the heading (heading *Zoos* contains a *see-also*-reference to *Tourist Attractions*).

The content of the headings is extremely rich in information especially because there are very few cases where a heading consists of more than 4 words. An index is created for nouns and noun phrases found in every heading. These are the index terms which we treat just like concepts in the *UKSearch* system. We treat two such concepts as related concepts if they were either extracted from the same heading or from two separate headings which are linked by a *see*-reference. We ignore *see-also*-references in the domain model construction process altogether. We then build a model as outlined in Sect. 4.1. A little technical note: for the YPA we do not build the model in an offline process, instead we perform these steps on the fly without database access (by simply consulting the concepts and relations between them).

We will see how this model is exploited in the dialogue to construct potential query modification options.

8.3 Dialogue Strategy in the YPA

A *goal description* in the YPA is more structured than in *UKSearch*.

The interaction between user and machine can be roughly summarized as a filling of different slots in a *slot-and-filler* query with the information extracted from the user input, so that the database access finally retrieves an acceptable set of entries. These slots represent different types of information such as *goods and services* or *location* information. The *slot-and-filler* query is encoded as part of the *goal description* which contains a number of other

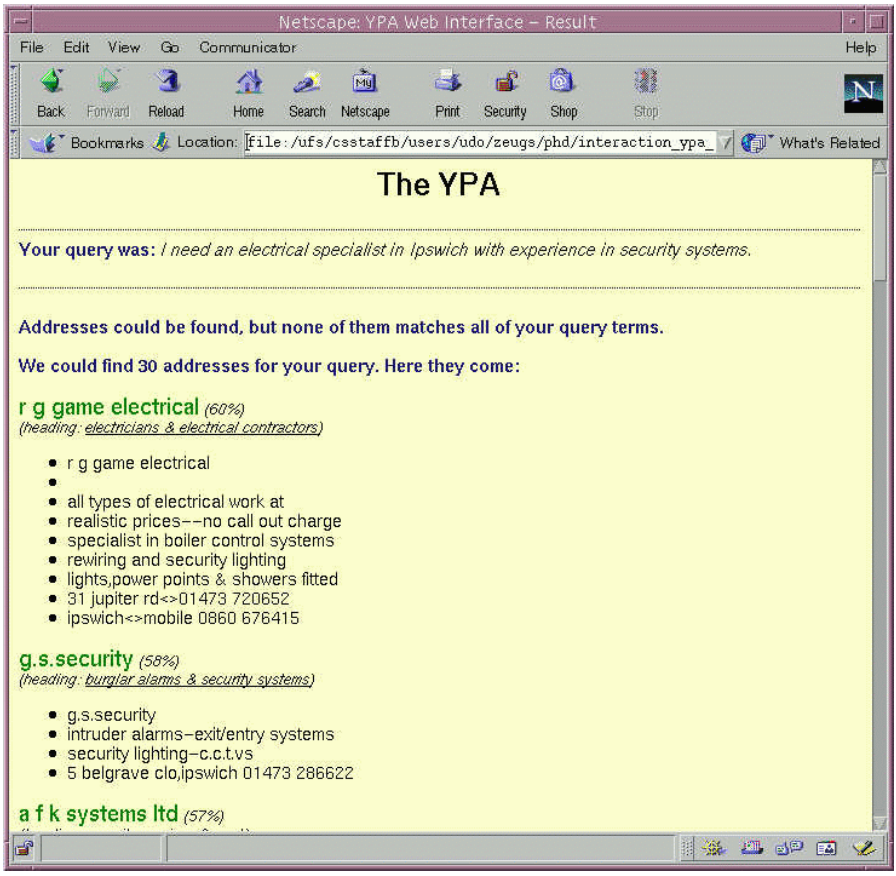

Fig. 8.3. YPA: response to user query *"I need an electrical specialist ..."*

parameters apart from the actual user request. In this process, the *Dialogue Manager* is the vital control module. Every time the *Dialogue Manager* is called it performs the following tasks:

- calling the *Natural Language Frontend*
- evaluating the parsed input and determining the dialogue state
- performing all actions corresponding to the state transition.

Conceptually the dialogue manager is no different to the one used in *UK-Search*. However, it is also more customized and capable of dealing with the more specific dialogue states that can occur in this application. A query for which the YPA can find a set of matching advertisements is displayed in Fig. 8.3.

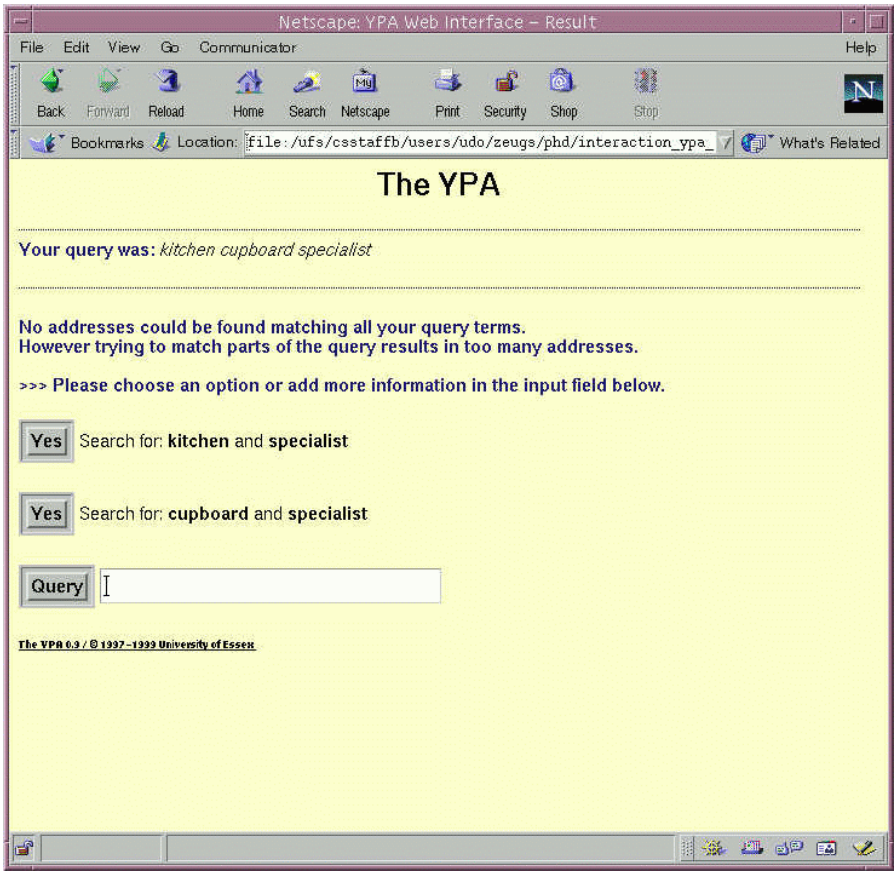

Fig. 8.4. YPA: response to user query *"kitchen cupboard specialist"*

For a second example see Fig. 8.4. Displayed is the response following the user query *"kitchen cupboard specialist"*. In this case the YPA was unable to find any matching advertisements and some query relaxations are suggested.

The dialogue manager is an implementation of the system outlined in Sect. 5.6. Here we demonstrate how the parts of the dialogue manager are set up. In this applications the *Query Construction Component* is responsible for matching goal descriptions against the database of advertisements and for calculating potential query modifications.

If the *Query Construction Component* can match the goal description against a reasonably sized set of addresses (i.e. advertisements) in the *Back-end* database, then these matches will be displayed. No query modifications will be calculated. The administrator setup defines up to how many addresses would be displayed. If the database delivers too few matches, then a general strategy of successive relaxation is applied. Relaxation options (i.e. modified

goal descriptions) would for example be constructed by ignoring "less important" slots (e.g. *opening hours* as opposed to *goods and services*) or by suggesting partial matches (as displayed in Fig. 8.4). If too many addresses are retrieved, then the query will be further constrained by calculating potential query refinement options. The domain model (which is part of the *World Model* component of the YPA) is utilized in the same way as the domain model in the *UKSearch* system.

8.3.1 Properties

We distinguish a number of *document properties* to express the relevant information we extract from advertisements (i.e. documents). The database contains corresponding tables, so that a database query can be more complicated than just a simple index lookup. The user input is parsed, so that the query terms for each of these document properties can be extracted appropriately:

- Keyword x can be found in the *goods and services* section of an advertisement (the indexing step interprets all free text as *goods and services* information that has not been identified as belonging to one of the other types of information such as location).
- Keyword x can be found in the business classification (of an advert).
- Keyword x matches a business name.
- Keyword x is a location name matching the location of the business.
- Keyword x matches the opening hours of a business.
- Keyword x can be found in the payment methods.
- Keyword x matches a business classification c.

The very last property in this list can be used to restrict the search so that only documents that satisfy the user query *and* are listed under a specific business classification should be retrieved (e.g. find only those plumbers in London that are classified under *Boiler Cleaning*).

Each of the properties can be mapped to a particular slot in the *slot-and-filler* query (but goods and services information and classification information are both kept in the same slot).

Unlike in the *UKSearch* system we do have *system properties* whose setup can be modified by the user or proposed by the system. The properties we distinguish can be characterized by a set of questions:

- Should an external world model be applied?
- Should all index tables (including the business names) be searched? Or just the free text?
- Should only business names be searched?

These questions are encoded in the user interface, so that a user can decide whether the default values of the appropriate system properties should be modified. For example, one version of the YPA gives the user the following options (possible choices shown in parentheses):

- Apply world knowledge: (always / never / only if needed)
- Search space: (all index files / only company names / all files except company names).

The resulting *goal description* is obviously more complex than in a simple Web search application. The values in the attribute-value pairs are updated by either adding, deleting or replacing elements (apart from keywords we keep track of some syntactic information derived from the input as well as conjunctions and disjunctions), but also by setting particular values for the system properties.

Assuming the user has typed in *"cameras in Colchester"*, then we will get a goal description such as this one:

```
Goal = [document_keyword_goods({cameras}),
        document_keyword_classification({}),
        document_name({}),
        document_location({colchester}),
        document_opening_hours({}),
        document_payment_method({}),
        document_classification({}),
        system_searchspace(default),
        system_worldmodel(off)]
```

8.3.2 Dialogue Setup

The *input* is either typed input or a selection of a tick box that is associated with a potential choice offered by the dialogue system.

The *dialogue history* contains the last query. This structure is only used for display purposes.

The *currentquery* function turns a goal description into a database query. Each slot corresponds to a particular set of index tables. The content of each slot is mapped to a query containing flat query terms in conjunctive normal form. The values of the system properties influence the query construction.

The *retrieve* function matches a query to a set of documents retrieved from the database following a query submission.

Again, the central dialogue state is the *Display* state (to be precise, the *Display* state is a generalization of three distinct states as explained in Sect. 5.6: a successful database access, *too many* matches and *too few* matches). Apart from that we have all the other states characterized in Chap. 5.6, namely:

- *Start* state
- *Meta* state
- *Unknown Input* state
- *Missing* state and
- *Inconsistency* state.

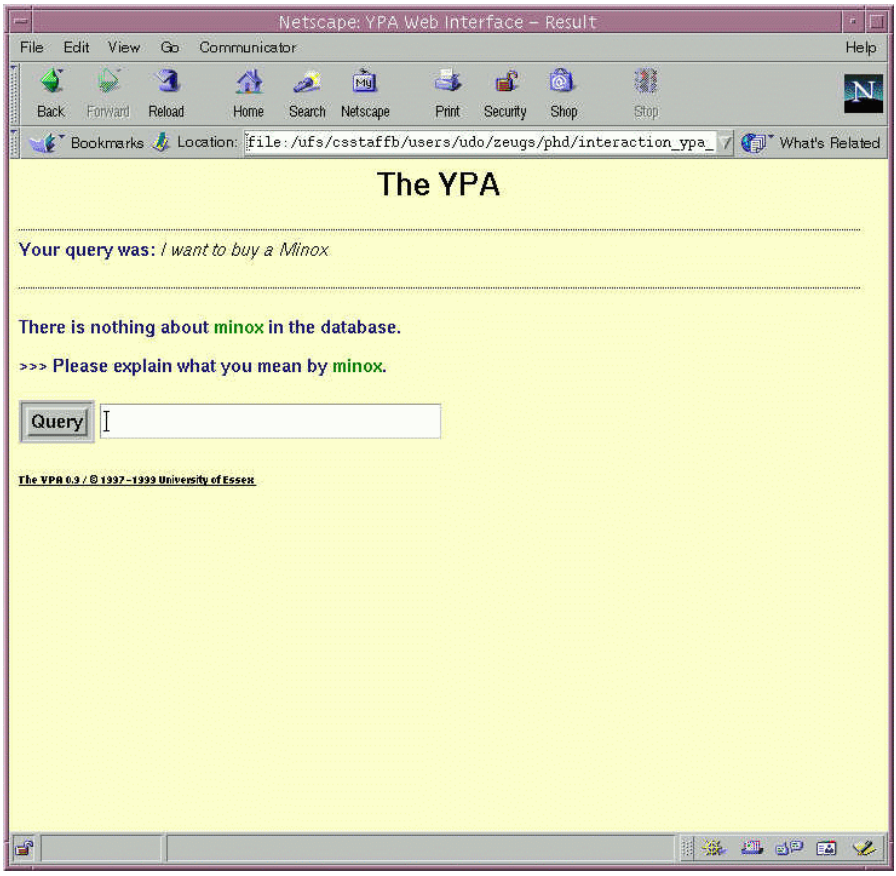

Fig. 8.5. YPA: response to user query *"I want to buy a Minox"*

The input parser detects requests for restarting the dialogue (i.e. a request to move to the *Start* state) or if the user asks for help (i.e. *Meta* state, which is treated just like the *Meta* state in the *UKSearch* system).

If the user input cannot be matched against the database, then the dialogue moves to the *Unknown Input* state. This triggers a clarification step and as a result of that the dialogue manager will either go into the *Display* state, remain in the *Unknown Input* state or go into one of the other states. The move into the *Display* state means that the goal description will be updated accordingly (the original query terms will be replaced by the newly input information). See Fig. 8.5 for an example input (*"I want to buy a Minox"*) that makes the system go into the *Unknown Input* state.

The information collected in a clarification step is logged and can be edited by an administrator or automatically added to the world model applied by

the YPA (i.e. a user input *"a camera"* results in a relation between the terms *minox* and *camera*).

The *Missing* state is reached if the *slot-and-filler* query contains unfilled slots that are required to be filled before the database can be queried (in the existing setup this is the case for the *goods and services* slot). Following a new user input the dialogue manager could again move to any of the states.

The *Inconsistency* state is reached when a technical problem occurs. This is a final state. The dialogue manager usually moves into this state when the database has been shut down (or updated without restarting the YPA).

A state diagram would contain transitions from any one state to any other. An exception is the *Inconsistency* state, which is a final state with no transitions to other states. But remember, that the user is free to add information at any stage in the dialogue and is not restricted to choosing one of the suggestions proposed by the system. Instead, the user could input some correction (e.g. *"no, I want an emergency dentist"*) or a request to restart (e.g. *"restart system!"*). Because we want to allow such input, we do not exclude any particular transition between high level dialogue states per se.

8.3.3 Dialogue Function

The dialogue function *trans* which maps a dialogue state and the user input to a new dialogue state does not differ in principle from the *UKSearch* system. The type of input expected from the user is exactly the same, either typed input or a selection.

- If the user *types* an input, then the goal description is updated accordingly by parsing the input and updating the *slot-and-filler* query. The new dialogue state is calculated by selecting potential choices for the new goal description (see below) and applying the necessary query and retrieve functions.
- If the user *selects* a potential choice $< Goal, Hist, Input >$, then *Goal* is the new goal description. This is equivalent to *UKSearch*.

Apart from the user input, the user can select system properties from lists of predefined options (e.g. to force the system to use synonyms from *WordNet* for query expansion etc.). These system properties are submitted together with the input and integrated in the new goal description accordingly.

8.3.4 Calculation of Potential Choices

This is where there are quite obvious differences between *UKSearch* and the YPA. *UKSearch* simply matches the input against a domain model. In the YPA on the other hand we have (apart from the domain model) a structured database and make assumptions about the types of information stored in that database.

In *UKSearch* we had a dialogue manager that had no direct access to the actual database of documents that the user searched. Instead a search engine was applied, and the dialogue manager constructed potential query modifications based on the knowledge encoded in the domain model. Here the situation is different. The dialogue manager can actually query the database and assess the effects of certain modifications to the result set. The potential choices that are to be presented to the user reflect much more what the current database has to offer. For example, before offering a choice to the user the dialogue manager can submit the new query and see whether the size of the result set would be acceptable.

Therefore, the calculation of the following steps that will lead to a set of potential choices might in fact involve several internal calls to the database:

- Calculate appropriate modifications of the document properties.
- Calculate appropriate modifications of the system properties.
- Rank all query and system modifications.
- Select the highest ranked modifications and construct potential choices.

Unlike the *UKSearch* system, the YPA is by default set up to not present addresses unless the query is specific enough or one of the presented options has been ticked. The administrator sets two thresholds max and min, where max is the maximum number of matches to be displayed (we use $max = 50$) and min is the minimum number of matches (we use $min = 1$). If there are more than max matching documents for the current goal description, the system will construct query refinements only. If there are less than min matches, the system tries to find query relaxations.

Based on our experience we perform only simple relaxations or refinements in order to not confuse the user, e.g. if the query consisted of two terms we would not construct a query modification that is based on applying query expansion to both query terms.

The query modification process is more customized than in the *UKSearch* system due to the fact that we can make more assumptions about the documents (i.e. advertisements), so that our index tables are *knowledge-rich* in the sense that they represent different aspects of a document such as location information, payment methods etc. Each of these types of information can be treated differently. The refinements and relaxations that we construct as potential query modifications apply this additional knowledge.

Potential query refinements are constructed by exploring each one of the following steps:

- Propose a single query term to be added to a particular slot (using the domain model in the same way as we have seen it in *UKSearch*, i.e. in the simplest case find a root node and propose the concepts in the daughter nodes as additional query terms). The term is simply added to the existing query encoded in the goal description. We do this only for the *goods and services* slot.

- Change a single system property (e.g. constraining the use of the world model(s), reducing the number of database tables to be accessed etc.).
- Select a single classification if all the query terms in the *goods and services* slot match the classification (i.e. if all query terms have been identified as concepts for a particular classification). The goal description is being changed by adding the appropriate document property (the new query matches all those documents that the original query matched and which are classified under a specific classification). For example, the user might have asked for *"alarms in Ipswich"*. A large number of advertisers exist, classified under different business classifications. The system therefore suggests that the user selects a specific classification (e.g. *"Burglar alarms & security systems"*, *"Car alarms & security"*, *"Fire alarms"* etc.) to narrow down the query.
- Check whether the *location* slot is empty. If so, the goal description will not be changed. However, the dialogue manager is set up to request location information from the user. The user is not required to follow this request. (The dialogue manager can be set up to allow a number of possible slots to be handled similarly.)

Potential query relaxations are constructed by exploring each one of the following steps:

- Relax a single slot. That could mean that the content is ignored altogether (e.g. propose matches that do not match the *opening hours* but everything else), or it could be based on additional world knowledge as it is used for the location information (e.g. propose the use of *essex* instead of *colchester*). All additional knowledge is incorporated in the *World Model* component of the YPA.
- Change a single system property (e.g. relaxing the use of the world model(s), expanding the number of database tables to be accessed etc.).
- Propose matches for partial queries (an example of which is shown in Fig. 8.4).
- Propose a single query term to be replaced by a disjunction of terms (using the synonyms in *WordNet*).
- Create a new goal description by replacing the content of the *goods and services* slot by a single noun phrase selected from this slot (everything else remains unchanged). Since we keep track of simple syntactic structures in the user input (mainly noun phrases), we are able to extract individual nouns or noun phrases from a slot.

The function *choicerank* ranks all query modifications that have been considered as potential choices (utilizing the ranking tables stored in the *Ranking Component*). The highest ranked choices are presented to the user (we use a maximum number $max = 10$). The *Backend* has detailed index tables that allow the YPA to assess what effect on the number of matches a particular query modification would have.

The potential choices are then put together in a straightforward way and passed to the dialogue manager to be encoded in the new dialogue state. It has already been mentioned that the user just has to tick a box, and behind the scenes the goal description is updated accordingly.

8.4 Implementational Issues

For the YPA we were given the source data in electronic format. This data is processed and stored in an *Oracle* database.

The Web-based *online* dialogue system runs as a *Sicstus* Prolog executable accessed via sockets. Perl scripts pass the user requests from the browser to the *Sicstus* system and return the answer once the system is finished. The robot as well as most of the index construction programming is done in Perl making use of existing modules (LWP, HTML, URI, Text etc.). For the indexing process we also use the Brill tagger [18]. The online system uses the C interface to *WordNet*.

9

Future Directions and Conclusions

The future directions will focus on three areas which will be discussed in this chapter:

- *Automatic adjustment of the domain model.* We suggest the exploration of ideas from collaborative filtering, i.e. logging a user's search behaviour will propose clues about how to update the concept hierarchies automatically so that the system can present better suggestions when the next user comes along.
- *Dialogue management.* It needs to be investigated how the presentation of query modification suggestions can be optimized. This touches human computer interaction issues.
- *Evaluation.* One interesting aspect to be investigated (and evaluated) in the future is to find out how the methods introduced in this book can be combined with other approaches.

We also see a possible application in the semi-automatic creation of term hierarchies where the documents are processed as described and the resulting domain models are then manually refined using appropriate tools.

Apart from that we need to stress, that our *UKSearch* experiments have so far only involved HTML documents, but for non-HTML documents it should be possible to use similar approaches as long as some layout analysis is applied which preprocesses the documents in order to mark up the various types of layout found in the documents.

9.1 Towards Evolving Domain Models

The task-based evaluations show that users strongly prefer a search system which incorporates automatically constructed domain knowledge over a standard search engine. Nevertheless, although users in our experiments expressed that they want some guidance in the search process, they also complained about some query modification terms that are misleading or not very useful.

Automatically extracted knowledge has its advantages, the main one is that such knowledge can be constructed rapidly. But the quality is not as good as if it had been customized manually. As with any statistical approach the model can only be as good as the data it is derived from. Naturally, there are flaws. The two obvious ways around this would have been to manually adjust the domain model, which is very expensive, or to apply user feedback which has been shown to be not very effective since most users simply ignore it (see for example [131]). A sensible approach seems to be a heterogeneous one which allows user feedback but does not rely on it. We see the main future direction in collaborative filtering techniques. Those are techniques in which the behaviour of a user is observed by the system in order to assist the next user with similar requests. One motivation is that "on intranets there is much to be gained by optimizing the search engine, perhaps via feedback learning, to accurately answer the top few queries" [48]. We propose to monitor the user dialogues and adjust the domain model accordingly. Simply doing this for every user individually seems to be of little use since we assume that even very frequent user queries are hardly ever submitted twice by the same user. Therefore we suggest to learn from a user in order to help the next user with a similar request. Furthermore, unlike in classical collaborative filtering this approach does not distinguish a number of user groups. We basically have one large group of users, those who submit queries to the search engine of the particular site. Thus, the idea is to improve the domain model of that site rather than user profiles. This section presents some preliminary ideas that will hopefully advance domain models such as those applied by the *UKSearch* system.

Let us assume, a user submits a query to *UKSearch* running on the Essex domain. The main focus shall be on that type of queries that triggers the system to present query refinements. In order to get to a more specific query, the dialogue component determines the "best" refinement terms and asks the user to choose one. To use our running example, a query for *"union"* would trigger the dialogue system to offer the listed choices, i.e. *students_union*, *european_union*, *christian_union* and *trade*. Any of those terms would constrain the original query and result in a much smaller set of matching documents. Whatever the user selects, the action is recorded and used to adjust the weights associated with particular concept terms and relations between those concepts in the domain model.

This differs from *explicit relevance feedback* [155, 23] in that the user is not required to judge the relevance of query modification options or documents but instead the user's selections are being observed. The idea is similar to the one described in [165] where *implicit relevance feedback* is used to update the result set in an ad hoc search situation. If a user moves the mouse over a document title, then it is interpreted as an expression of interest in the particular document. However, here we only look at the actual user selections to update the domain model.

In that sense, we can see the outlined approach as a particular application of collaborative filtering. Collaborative filtering is based on identifying the opinions and preferences of similar users in order to predict the preferences and to recommend items to others as used in systems to recommend news (e.g. *GroupLens* [126]) or movies (e.g. *Video Recommender* [69]). Collaborative search has also been proposed for using similar past queries to automatically expand new queries, e.g. [52] and [124] where documents that correspond well to similar queries also provide feedback on the original query.

In our search framework the domain model is used to suggest a selection of query refinement terms if the query matches a large number of documents.

Now assume, for every concept in the domain model the weights associated with the branches originating in a node are equal and sum up to 1. These weights change, if:

- a concept in a daughter node is offered and selected by the user (increase)
- a concept in a daughter node is offered and not selected (decrease).

This will only change weights of relations already in place. The result is that the good parts of the domain model will gain importance, the rest will be less and less relevant. But that does not allow the creation of new links. However, imagine in our example query (*"union"*) the user ignores all the options and inputs *"students union societies"*. This user will implicitly introduce a link between *union* and *students_union_societies*. This link will gain importance if this sequence of user inputs is observed a number of times. Thresholds could then be introduced to delete nodes from a hierarchy in the domain model or to include new ones.

To make the ideas a bit more concrete we shall look at Fig. 9.1 which displays one hierarchy of the originally constructed domain model (we use single concepts to represent a node in the model).

While the domain model essentially remains as it is, the weights are adjusted over time and after a trial period the user would get a different picture. Figure 9.2 shows how the encoded relations may have changed over time. For example, *christian_union* does not seem to have attracted any interest. The weight dropped so that this concept is no longer part of the hierarchy. However, entering the phrase as a query could bring it back. Moreover, the term *welfare* which refers to "welfare services" offered by the students' union has become more important and is offered as a possible choice following the user query *"union"*. The two terms *union* and *welfare* have always been related concepts, but the relevance of this branch compared to the other four displayed in Fig. 9.1 was too low to be displayed in the original setup.

Note that the outlined system setup does not restrict us to any domain knowledge construction method in particular. The automated adjustment process is generic and one could easily substitute the actual model with any domain model that has a similar structure, e.g. [134, 9, 99].

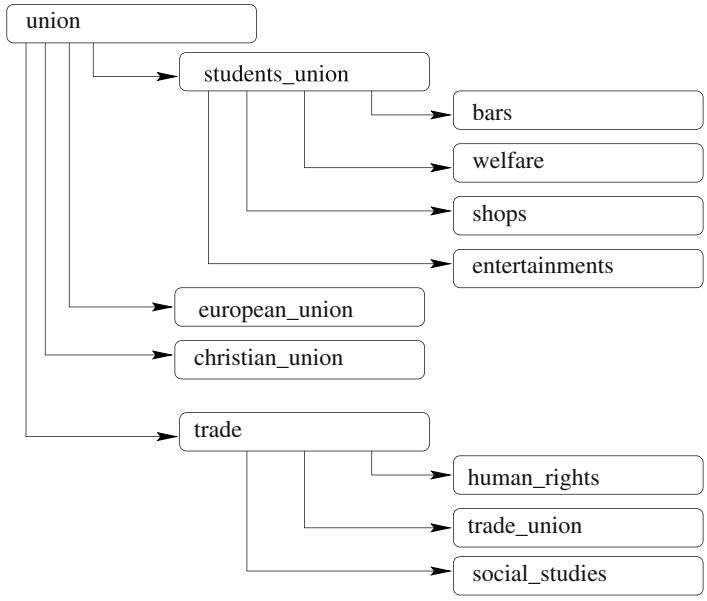

Fig. 9.1. Original concept tree for example query *"union"*

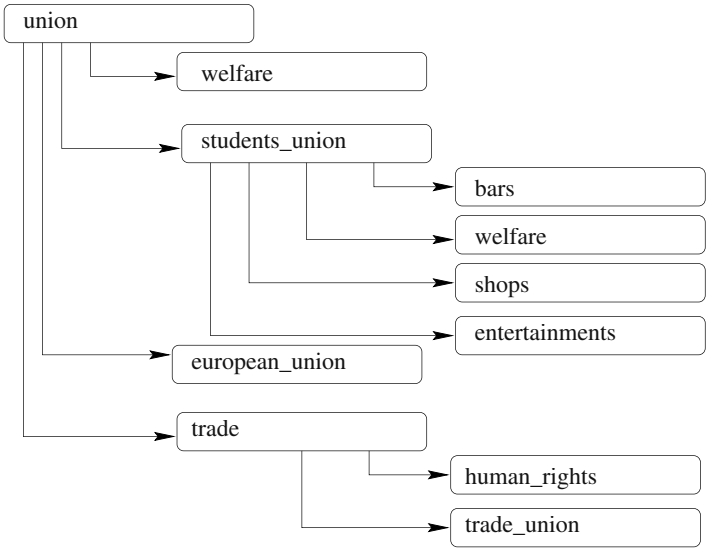

Fig. 9.2. Concept tree for example query *"union"* after trial period

9.2 Dialogue Management

Currently, the dialogue system presents the user with a flat list of concepts that can serve as query modification terms. Those concepts are extracted from

the automatically constructed domain models using some ranking function. However, to support the user in the process of *exploring* the document collection one could think of a much more advanced interface that presents the concepts in a wider context, similar to the graph representation in the DOPE browser [148], in TouchGraph's[1] visual browsers or the type of semantic graphs used in the Infomap Project[2] to visualize words and their relationships.

In this context it would mean, instead of selecting single concepts one might present entire hierarchies so that a user can navigate through some active map of linked concepts. The "current position" in the hierarchies would need to be highlighted. Different hierarchies of the same domain model could be linked by nodes that represent the same sets of concepts. One could avoid some of the problems involved in selecting the best query modifications by letting the user decide. However, this is quite a complex area which requires careful consideration of human computer interaction aspects. Therefore it has been left as a future issue.

Obviously, research into dialogue management is not restricted to our simplistic view of what a dialogue manager should be capable of and what we understand as "dialogue" in this context. There are numerous research directions and we presented a narrow picture that seems sufficient for our task, the search in collections of partially structured documents.

9.3 An Outlook on Future Evaluations

We have learned a lot about advantages and disadvantages of the techniques introduced in this book in a set of controlled experiments. The ultimate test of these extraction and search methods will be large scale experiments involving real users on some permanently installed search engines. However, the difficulty is to compare the results with alternative approaches. For any future comparison between alternative techniques we see the TREC interactive track framework model as a sensible approach. The principles as outlined in [67] are useful for any such future evaluations too:

- an interactive search task - question answering
- 8 questions - short answers
- 16 searchers - minimum
- a newswire/newspaper collection to be searched
- a required set of questionnaires.

Future task-based evaluations will basically follow the procedure that has been used to evaluate *UKSearch* for the Essex and BBC News domains. In addition to the document collections we would need a corpus of existing user

[1] `http://www.touchgraph.com`

[2] `http://infomap.stanford.edu/graphs/`

queries. A useful source would be the log files of queries submitted through an existing Web search or intranet search engine at the particular site.

However, other experiments will need to address the issues of presenting search options and search results in a graphical user interface. We found evidence for and against the fact that a simple search interface might be prefered to one that displays options alongside search results. But a more sophisticated interface will make this an even more complex task. Here we enter (again) the research area of human computer interaction which will have to answer the question of how to get measurable results concerning user perception and preferences.

9.4 Conclusions

Search engine technology is an exciting research area. We simply picked one aspect from this rather large field. We started by discussing some common problems that arise when searching document collections. We focused on such document collections that can be characterized as *partially structured*. Examples are Web sites or intranets.

Such document collections can vary a lot in size and how much structure they carry. What they have in common is that they typically do have *some* structure and that they cover a limited range of topics. The second point is significantly different from the *Web* in general.

The presented work aims at assisting a user in the search process so that information can be located much easier. The type of search system that we propose goes beyond a standard search engine. Apart from displaying the best matches in some ranked order it can also suggest ways of refining or relaxing the query. We guide a user through the information available.

In order to suggest sensible query modifications we would need to *know* what the documents are about. Explicit knowledge about the document collection that has been encoded in some electronic form is what we need. However, typically such knowledge is not available. So we decided to construct it automatically.

How can this be achieved? A tremendous amount of implicit knowledge is stored in the *markup* of documents. But not much has been done to use this particular knowledge. This book described a method of automatically extracting such knowledge and organizing it in a *domain model*. It also introduced a simple dialogue system that can apply the extracted knowledge as part of an online search system.

The book has demonstrated three main aspects:

- We have shown how markup structure can be used to build domain models quickly and fully automatically.
- We have shown how to apply the knowledge that has been extracted in the domain model construction step. It was demonstrated how a generic

dialogue manager can utilize such knowledge to assist a user by refining the choices as he or she searches the document collection.

- Two implemented search systems - *UKSearch* and the YPA - have demonstrated the usefulness of this approach. The systems have been implemented to work on different document collections. They focused on two aspects: *UKSearch* demonstrated the use of the automatically constructed domain models in the search process. The YPA system (which was discussed more briefly) was more concerned with the dialogue system. We have also presented detailed evaluation results for *UKSearch*.

With the methods introduced in this book it should be possible to build dialogue-based search systems rapidly and fully automatically for a variety of partially structured document collections. Nevertheless, there is plenty of scope for further research.

References

1. S. Abiteboul. Querying Semi-Structured Data (invited talk). In *Proceedings of the 6^{th} International Conference on Database Theory (ICDT)*, pages 1–18, Delphi, Greece, 1997.
2. R. Agarwal. Towards a PURE Spoken Dialogue System for Information Access. In *Proceedings of the ACL/EACL Workshop on "Interactive Spoken Dialog Systems: Bringing Speech and NLP Together in Real Applications"*, pages 90–97, Madrid, 1997.
3. K. Ahmad, M. Tariq, B. Vrusias, and C. Handy. Corpus-Based Thesaurus Construction for Image Retrieval in Specialist Domains. In F. Sebastiani, editor, *Proceedings of the 25^{th} European Colloquium on Information Retrieval Research (ECIR'03)*, Lecture Notes in Computer Science 2633, pages 502–510, Pisa, 2003. Springer Verlag.
4. H. Alani, S. Kim, D. E. Millard, M. J. Weal, W. Hall, P. H. Lewis, and N. R. Shadbolt. Automatic Ontology-Based Knowledge Extraction from Web Documents. *IEEE Intelligent Systems*, 18(1):14–21, January/February 2003.
5. J. Allen, L. Schubert, G. Ferguson, P. Heeman, C. Hwang, T. Kato, M. Light, N. Martin, B. Miller, M. Poesio, and D. Traum. The TRAINS project: a case study in building a conversational planning agent. *Journal of Experimental and Theoretical Artificial Intelligence*, 7:7–48, 1995.
6. E. Amitay. Using common hypertext links to identify the best phrasal description of target web documents. In *Proceedings of the SIGIR'98 Post-Conference Workshop on Hypertext Information Retrieval for the Web*, Melbourne, 1998.
7. P. Anick. Using Terminological Feedback for Web Search Refinement - A Log-based Study. In *Proceedings of the 26^{th} Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 88–95, Toronto, Canada, 2003.
8. P. G. Anick. *Automatic Construction of Faceted Terminological Feedback for Context-Based Information Retrieval*. PhD thesis, Brandeis University, 1999.
9. P. G. Anick and S. Tipirneni. The paraphrase search assistant: terminological feedback for iterative information seeking. In *Proceedings of the 22^{nd} Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 153–159, Berkeley, CA, 1999.

10. G. Attardi, A. Gullí, and F. Sebastiani. Automatic Web page categorization by link and context analysis. In C. Hutchison and G. Lanzarone, editors, *Proceedings of THAI-99, European Symposium on Telematics, Hypermedia and Artificial Intelligence*, pages 105–119, Varese, Italy, 1999.
11. H. Aust, M. Oerder, F. Seide, and V. Steinbiss. The Philips automatic train timetable information system. *Speech Communication*, 17:249–262, 1995.
12. R. Baeza-Yates and B. Ribeiro-Neto. *Modern Information Retrieval*. Addison Wesley, 1999.
13. M. Beaulieu, H. Fowkes, N. Alemayehu, and M. Sanderson. Interactive Okapi at Sheffield - TREC-8. In *Proceedings of the Eighth Text Retrieval Conference (TREC-8)*, pages 689–698, NIST Special Publication 500-246, 1999.
14. S. M. Beitzel, E. C. Jensen, A. Chowdhury, D. Grossman, and O. Frieder. Hourly Analysis of a Very Large Topically Categorized Web Query Log. In *Proceedings of the 27^{th} Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 321–328, Sheffield, 2004.
15. D. J. Bell and I. Ruthven. Searcher's Assessments of Task Complexity for Web Searching. In *Proceedings of the 26^{th} European Conference on Information Retrieval (ECIR'04)*, Lecture Notes in Computer Science, pages 57–71, Sunderland, 2004. Springer Verlag.
16. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 5:34–43, May 2001.
17. T. Bray, J. Paoli, and C. Sperberg-McQueen. Extensible markup language (XML) 1.0. W3C Recommendation, 1998.
18. E. Brill. A simple rule-based part of speech tagger. In *Proceedings of the Third Conference on Applied Natural Language Processing, ACL*, pages 152–155, Trento, Italy, 1992.
19. E. Brill. Some advances in rule-based part of speech tagging. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94)*, pages 722–727, Seattle, Wa., 1994.
20. S. Brin and L. Page. The Anatomy of a Large-Scale Hypertextual Web Search Engine. In *Proceedings of the Seventh International World Wide Web Conference (WWW7)*, pages 107–117, Brisbane, 1998.
21. A. Broder. A Taxonomy of Web Search. *SIGIR Forum*, 36(2):3–10, 2002.
22. P. Bruza, R. McArthur, and S. Dennis. Interactive Internet search: keyword, directory and query reformulation mechanisms compared. In *Proceedings of the 23^{rd} Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 280–287, Athens, Greece, 2000.
23. C. Buckley, G. Salton, and J. Allan. The effect of adding relevance information in a relevance feedback environment. In *Proceedings of the 17^{th} Annual International ACM SIGIR Conference on Research and Development in Information*, pages 292–301, 1994.
24. P. Buneman. Semistructured Data (invited tutorial). In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, pages 117–121, Tucson, Arizona, 1997.
25. C. Carson, S. Belongie, H. Greenspan, and J. Malik. Blobworld: Image Segmentation Using Expectation-Maximization and Its Application to Image Querying. *IEEE Transactions on Analysis and Machine Intelligence*, 24(8):1026–1038, 2002.

26. J. Chai, V. Horvath, N. Nicolov, M. Stys, N. Kambhatla, W. Zadrozny, and P. Melville. Natural Language Assistant - A Dialog System for Online Product Recommendation. *AI Magazine*, 23(2):63–75, 2002.
27. J. Chai, J. Lin, W. Zadrozny, T. Ye, M. Stys-Budzikowska, V. Horvath, N. Kambhatla, and C. Wolf. The Role of a Natural Language Conversational Interface in Online Sales: A Case Study. *International Journal of Speech Technology*, 4:285–295, 2001.
28. S. Chakrabarti. *Mining the Web - Discovering Knowledge from Hypertext Data*. Morgan Kaufmann, 2003.
29. S. Chakrabarti, B. Dom, D. Gibson, J. Kleinberg, S.R. Kumar, P. Raghavan, S. Rajagopalan, and A. Tomkins. Hypersearching the Web. *Scientific American*, 6:54–60, June 1999.
30. S. Chakrabarti, B. Dom, P. Raghavan, S. Rajagopalan, D. Gibson, and J. Kleinberg. Automatic Resource Compilation by Analyzing Hyperlink Structure and Associated Text. In *Proceedings of the Seventh International World Wide Web Conference (WWW7)*, pages 65–74, Brisbane, 1998.
31. C. H. Chang and C. C. Hsu. Enabling Concept-Based Relevance Feedback for Information Retrieval on the WWW. *IEEE Transactions on Knowledge and Data Engineering*, 11(4):595–609, July/August 1999.
32. H. Chen and S. T. Dumais. Bringing order to the web: Automatically categorizing search results. In *Proceedings of CHI'00, Human Factors in Computing Systems*, pages 145–152, Den Haag, 2000.
33. M. Chen, M. Hearst, J. Hong, and J. Lin. Cha-Cha: A System for Organizing Intranet Search Results. In *Proceedings of the 2^{nd} USENIX Symposium on Internet Technologies and Systems (USITS)*, pages 47–58, Boulder, CO, 1999.
34. S.-L. Chuang and L.-F. Chien. Enriching Web taxonomies through subject categorization of query terms from search engine logs. *Decision Support Systems*, 35:113–127, 2003.
35. F. Ciravegna, A. Dingli, D. Guthrie, and Y. Wilks. Integrating Information to Bootstrap Information Extraction from Web Sites. In *Proceedings of the IJCAI-03 Workshop on Information Integration on the Web*, pages 9–14, Acapulco, 2003.
36. J. Cowie and Y. Wilks. Information Extraction. In R. Dale, H. Moisl, and H. Somers, editors, *Handbook of Natural Language Processing*, pages 241–260. Marcel Dekker, New York, 2000.
37. N. Craswell, D. Hawking, J. Thom, T. Upstill, R. Wilkinson, and M. Wu. TREC11 Web and Interactive Tracks at CSIRO. In *Proceedings of the Eleventh Text Retrieval Conference (TREC-2002)*, NIST Special Publication 500-251, 2003.
38. N. Craswell, D. Hawking, R. Wilkinson, and M. Wu. TREC10 Web and Interactive Tracks at CSIRO. In *Proceedings of the Tenth Text Retrieval Conference (TREC-2001)*, pages 151–158, NIST Special Publication 500-250, 2002.
39. N. Craswell, D. Hawking, R. Wilkinson, and M. Wu. Overview of the TREC 2003 Web Track. In *Proceedings of the Twelfth Text Retrieval Conference (TREC 2003)*, pages 78–92, NIST Special Publication 500-255, 2004.
40. H. Cunningham. Information Extraction - a User Guide. Research memo CS-99-07, Institute for Language, Speech and Hearing (ILASH), and Department of Computer Science, University of Sheffield, 1999.

41. J. R. Curran and R. K. Wang. Transformation-Based Learning for Automatic Translation from HTML to XML. In *Proceedings of the Fourth Australasian Document Computing Symposium*, Coffs Harbour, Australia, 1999.
42. D. R. Cutting, D. R. Karger, J. O. Pedersen, and J. W. Tukey. Scatter/Gather: A Cluster-based Approach to Browsing Large Document Collections. In *Proceedings of the Fifteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR'92)*, pages 318–329, Copenhagen, Denmark, 1992.
43. A. De Roeck, U. Kruschwitz, P. Neal, P. Scott, S. Steel, R. Turner, and N. Webb. YPA - an intelligent directory enquiry assistant. *BT Technology Journal*, 16(3):145–155, 1998.
44. A. De Roeck, U. Kruschwitz, P. Scott, S. Steel, R. Turner, and N. Webb. The YPA - An Assistant for Classified Directory Enquiries. In B. Azvine, N. Azarmi, and D. Nauck, editors, *Intelligent Systems and Soft Computing: Prospects, Tools and Applications*, Lecture Notes in Artificial Intelligence 1804, pages 239–258. Springer Verlag, 2000.
45. E. Desmontils and C. Jacquin. Indexing a Web Site with a Terminology Oriented Ontology. In *Proceedings of the Semantic Web Working Symposium (SWWS'2001)*, pages 549–565, Stanford, 2001.
46. S. Dumais and H. Chen. Hierarchical Classification of Web Content. In *Proceedings of the 23^{rd} Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 256–263, Athens, Greece, 2000.
47. N. Eiron and K. S. McCurley. Analysis of anchor text for web search. In *Proceedings of the 26^{th} Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 459–460, Toronto, Canada, 2003.
48. R. Fagin, R. Kumar, K. McCurley, J. Novak, D. Sivakumar, J. A. Tomlin, and D. P. Williamson. Searching the Workplace Web. In *Proceedings of the Twelfth International World Wide Web Conference (WWW2003)*, pages 366–375, Budapest, 2003.
49. M. Fasli and U. Kruschwitz. Using Implicit Relevance Feedback in a Web Search Assistant. In N. Zhong, Y. Yao, J. Liu, and S. Ohsuga, editors, *Web Intelligence: Research and Development*, Lecture Notes in Artificial Intelligence 2198, pages 356–360. Springer Verlag, 2001.
50. C. Fellbaum, editor. *WordNet: An Electronic Lexical Database*. MIT Press, 1998.
51. D. Fensel, F. van Harmelen, I. Horrocks, D. L. McGuinness, and P. F. Patel-Schneider. OIL: An Ontology Infrastructure for the Semantic Web. *IEEE Intelligent Systems*, 16(2):38–45, March/April 2001.
52. L. Fitzpatrick and M. Dent. Automatic Feedback Using Past Queries: Social Searching? In *Proceedings of the 20^{th} Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 306–313, Philadelphia, PA, 1997.
53. S. Flank. A layered approach to NLP-based Information Retrieval. In *Proceedings of the 36^{th} ACL and the 17^{th} COLING Conferences*, pages 397–403, Montreal, 1998.
54. A. Fujii and T. Ishikawa. Utilizing the World Wide Web as an Encyclopedia: Extracting Term Descriptions from Semi-Structured Texts. In *Proceedings of*

the 38th Annual Meeting of the Association for Computational Linguistics, pages 488–495, Hong Kong, 2000.
55. J. Fürnkranz. Exploiting Structural Information for Text Classification on the WWW. In *Proceedings of the 3rd Symposium on Intelligent Data Analysis (IDA-99)*, pages 487–498, Amsterdam, 1999. Springer Verlag.
56. A. Gangemi, N. Guarino, C. Masolo, and A. Oltramari. Sweetening WORDNET with DOLCE. *AI Magazine*, 24(3):13–24, 2003.
57. E. J. Glover, D. M. Pennock, S. Lawrence, and R. Krovetz. Inferring Hierarchical Descriptions. In *Proceedings of 2002 ACM CIKM International Conference on Information and Knowledge Management*, pages 507–514, McLean, Virginia, 2002.
58. E. J. Glover, K. Tsioutsiouliklis, S. Lawrence, D. M. Pennock, and G. W. Flake. Using Web Structure for Classifying and Describing Web Pages. In *Proceedings of the Eleventh International World Wide Web Conference (WWW2002)*, pages 562–569, Honolulu, 2002.
59. S. J. Green. Automated link generation: can we do better than term repetition? In *Proceedings of the Seventh International World Wide Web Conference (WWW7)*, pages 75–84, Brisbane, 1998.
60. N. Guarino, C. Masolo, and G. Vetere. OntoSeek: Content-Based Access to the Web. *IEEE Intelligent Systems*, 14(3):70–80, May/June 1999.
61. C. Gutwin, G. Paynter, I. Witten, C. Nevill-Manning, and E. Frank. Improving browsing in digital libraries with keyphrase indexes. *Decision Support Systems*, 27:81–104, 1999.
62. D. Hawking and N. Craswell. Overview of the TREC-2001 Web Track. In *Proceedings of the Tenth Text Retrieval Conference (TREC-2001)*, pages 61–67, NIST Special Publication 500-250, 2002.
63. D. Hawking, E. Voorhees, N. Craswell, and P. Bailey. Overview of the TREC-8 Web Track. In *Proceedings of the Eighth Text Retrieval Conference (TREC-8)*, pages 131–150, NIST Special Publication 500-246, 1999.
64. J. Heflin and J. Hendler. A Portrait of the Semantic Web in Action. *IEEE Intelligent Systems*, 16(2):54–59, March/April 2001.
65. M. R. Henzinger, R. Motwani, and C. Silverstein. Challenges in Web Search Engines. *SIGIR Forum*, 36(2):11–22, 2002.
66. W. Hersh. TREC 2002 Interactive Track Report. In *Proceedings of the Eleventh Text Retrieval Conference (TREC-2002)*, NIST Special Publication 500-251, 2003.
67. W. Hersh and P. Over. TREC-9 Interactive Track Report. In *Proceedings of the Ninth Text Retrieval Conference (TREC-9)*, pages 41–50, NIST Special Publication 500-249, 2001.
68. W. Hersh, L. Sacherek, and D. Olson. Observations of Searchers: OHSU TREC 2001 Interactive Track. In *Proceedings of the Tenth Text Retrieval Conference (TREC-2001)*, pages 434–441, NIST Special Publication 500-250, 2002.
69. W. Hill, L. Stead, M. Rosenstein, and G. Furnas. Recommending and evaluating choices in a virtual community of use. In *Proceedings of the Conference on Human Factors in Computing Systems CHI'95*, pages 194–201, New York, 1995. ACM.
70. J. Hodgson. Do HTML Tags Flag Semantic Content? *IEEE Internet Computing*, 5(1):20–25, January/February 2001.

71. B. Hyusein and A. Patel. Web Document Indexing and Retrieval. In A. F. Gelbukh, editor, *Proceedings of the 4^{th} International Conference of Computational Linguistics and Intelligent Text Processing (CICLing 2003)*, Lecture Notes in Computer Science 2588, pages 573–579, Mexico-City, 2003. Springer Verlag.
72. International Organization for Standardization. ISO/IEC 13250:2002 Topic Maps, 2002.
73. B. J. Jansen, J. Bateman, and T. Saracevic. Real life information retrieval: A study of user queries on the web. *SIGIR Forum*, 32(1):5–17, 1998.
74. H. Joho, C. Coverson, M. Sanderson, and M. Beaulieu. Hierarchical Presentation of Expansion Terms. In *Proceedings of ACM Symposium on Applied Computing (SAC'2002)*, pages 645–649, Madrid, 2002.
75. H. Joho, M. Sanderson, and M. Beaulieu. A Study of User Interaction with a Concept-based Interactive Query Expansion Support Tool. In *Proceedings of the 26^{th} European Conference on Information Retrieval (ECIR'04)*, Lecture Notes in Computer Science, pages 42–56, Sunderland, 2004. Springer Verlag.
76. J. S. Justeson and S. M. Katz. Technical terminology: some linguistic properties and an algorithm for identification in text. *Natural Language Engineering*, 1(1):9–27, 1995.
77. J. Karlgren. *Stylistic Experiments for Information Retrieval*. PhD thesis, Swedish Institute of Computer Science, 2000.
78. B. Katz, J. Lin, and S. Felshin. Gathering Knowledge for a Question Answering System from Heterogeneous Information Sources. In *Proceedings of the ACL/EACL Workshop on Human Language Technology and Knowledge Management*, Toulouse, 2001.
79. B. Katz, D. Yure, J. Lin, S. Felshin, R. Schulman, A. Ilik, A. Ibrahim, and P. Osafo-Kwaako. Integrating Web Resources and Lexicons into a Natural Language Query System. In *Proceedings of the International Conference on Multimedia Computing and Systems (IEEE ICMCS '99)*, pages 255–261, Florence, 1999.
80. S. M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Transactions on Acoustics, Speech and Signal Processing*, 35(3):400–401, 1987.
81. C.-P. Klas and N. Fuhr. A new Effective Approach for Categorizing Web Documents. In *Proceedings of the 22^{nd} BCS-IRSG 2000 Colloquium on IR Research*, Cambridge, 2000.
82. J. M. Kleinberg. Authoritative Sources in a Hyperlinked Environment. In *Proceedings of the 9^{th} ACM-SIAM Symposium on Discrete Algorithms*, pages 668–677. ACM, 1998.
83. C. A. Knoblock, J. L. Ambite, S. Minton, M. Kolahdouzan, M. Muslea, J. Oh, and S. Thakkar. Integrating the World: The WorldInfo Assistant. In *Proceedings of the International Conference on Artificial Intelligence (IC-AI)*, pages 1355–1361, Las Vegas, 2001.
84. C. A. Knoblock, S. Minton, J. L. Ambite, M. Muslea, J. Oh, and M. Frank. Mixed-Initiative, Multi-Source Information Assistants. In *Proceedings of the Tenth International World Wide Web Conference (WWW10)*, pages 697–707, Hong Kong, 2001.

85. R. Kraft and J. Zien. Mining Anchor Text for Query Refinement. In *Proceedings of the 13th International World Wide Web Conference (WWW2004)*, pages 666–674, New York, 2004.
86. U. Kruschwitz. UKSearch - Web Search with Knowledge-Rich Indices. In *Proceedings of the AAAI-2000 Workshop on Artificial Intelligence for Web Search*, Technical Report WS-00-01, pages 41–45, Austin, TX, 2000. AAAI Press.
87. U. Kruschwitz. A Rapidly Acquired Domain Model Derived from Markup Structure. In *Proceedings of the ESSLLI'01 Workshop on Semantic Knowledge Acquisition and Categorisation*, Helsinki, 2001.
88. U. Kruschwitz. Dialogue for Web Search Utilizing Automatically Acquired Domain Knowledge. In V. Matoušek, P. Mautner, R. Mouček, and K. Taušer, editors, *Text, Speech, and Dialogue. Fourth International Conference (TSD2001)*, Lecture Notes in Artificial Intelligence 2166, pages 365–372. Springer Verlag, 2001.
89. U. Kruschwitz. Exploiting Structure for Intelligent Web Search. In *Proceedings of the 34th Hawaii International Conference on System Sciences (HICSS)*, pages 1356–1364, Maui, Hawaii, 2001. IEEE.
90. U. Kruschwitz. An Adaptable Search System for Collections of Partially Structured Documents. *IEEE Intelligent Systems*, 18(4):44–52, July/August 2003.
91. U. Kruschwitz. Automatically Acquired Domain Knowledge for ad hoc Search: Evaluation Results. In *Proceedings of the 2003 International Conference on Natural Language Processing and Knowledge Engineering (NLP-KE'03)*, pages 525–532, Beijing, 2003. IEEE.
92. U. Kruschwitz and H. Al-Bakour. Users Want More Sophisticated Search Assistants - Results of a Task-Based Evaluation. *Journal of the American Society for Information Science and Technology (JASIST)*, 2005. To appear.
93. U. Kruschwitz, A. De Roeck, P. Scott, S. Steel, R. Turner, and N. Webb. Extracting Semistructured Data - Lessons Learnt. In *Natural Language Processing - NLP2000: Second International Conference*, Lecture Notes in Artificial Intelligence 1835, pages 406–417. Springer Verlag, 2000.
94. N. Kushmerick, D. Weld, and B. Doorenbos. Wrapper Induction for Information Extraction. In *Proceedings of IJCAI-97*, pages 729–735, Nagoya, 1997.
95. S. Larsson and D. Traum. Information state and dialogue management in the TRINDI Dialogue Move Engine Toolkit. *Natural Language Engineering*, 6(3-4):323–340, 2000. Special Issue on Best Practice in Spoken Language Dialogue Systems Engineering.
96. S. Lawrence, C. L. Giles, and K. Bollacker. Digital libraries and autonomous citation indexing. *IEEE Computer*, 32(6):67–71, 1999.
97. S. Lawrence and C. Lee Giles. Accessibility of information on the web. *Nature*, 400(July 8):107–109, 1999.
98. D. Lawrie and W. B. Croft. Discovering and Comparing Topic Hierarchies. In *Proceedings of RIAO'2000*, pages 314–330, Paris, 2000.
99. D. J. Lawrie and W. B. Croft. Generating Hierarchical Summaries for Web Searches. In *Proceedings of the 26th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 457–458, Toronto, Canada, 2003.

100. A. Y. Levy, A. Rajaraman, and J. J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In *Proceedings of the 22^{nd} VLDB Conference*, pages 251–262, Mumbai (Bombay), India, 1996.
101. D. D. Lewis and K. Spärck Jones. Natural language processing for information retrieval. *Communications of the ACM*, 39(1):92–101, 1996.
102. Y. Li. Toward a Qualitative Search Engine. *IEEE Internet Computing*, 2(4):24–29, July/August 1998.
103. B. Liu, C. W. Chin, and H. T. Ng. Mining Topic-Specific Concepts and Definitions on the Web. In *Proceedings of the Twelfth International World Wide Web Conference (WWW2003)*, pages 251–260, Budapest, 2003.
104. A. Maedche, B. Motik, L. Stojanovic, R. Studer, and R. Volz. An Infrastructure for Searching, Reusing and Evolving Distributed Ontologies. In *Proceedings of the Twelfth International World Wide Web Conference (WWW2003)*, pages 439–448, Budapest, 2003.
105. A. Maedche, B. Motik, L. Stojanovic, R. Studer, and R. Volz. Ontologies for Enterprise Knowledge Management. *IEEE Intelligent Systems*, 18(2):26–33, March/April 2003.
106. A. Maedche and S. Staab. Ontology Learning for the Semantic Web. *IEEE Intelligent Systems*, 16(2):72–79, March/April 2001.
107. M. Margennis and C. J. van Rijsbergen. The potential and actual effectiveness of interactive query expansion. In *Proceedings of the 20^{th} Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 324–332, Philadelphia, PA, 1997.
108. S. McGlashan, N. Fraser, N. Gilbert, E. Bilange, P. Heisterkamp, and N. Youd. Dialogue Management for Telephone Information Systems. In *Proceedings of the International Conference on Applied Language Processing*, pages 245–246, Trento, Italy, 1992.
109. J. McHugh, S. Abiteboul, R. Goldman, D. Quass, and J. Widom. Lore: A Database Management System for Semistructured Data. *SIGMOD Record*, 26(3):50–66, 1997.
110. M. Michalowski, J. L. Ambite, S. Thakkar, R. Tuchinda, C. A. Knoblock, and S. Minton. Retrieving and Semantically Integrating Heterogeneous Data from the Web. *IEEE Intelligent Systems*, 19(3):72–79, May/June 2004.
111. D. S. Modha and S. Spangler. Clustering hypertext with applications to web searching. In *Proceedings of ACM Hypertext Conference*, pages 143–152, San Antonio, TX, 2000.
112. D. Moldovan, R. Girju, and V. Rus. Domain-Specific Knowledge Acquisition from Text. In *Proceedings of the Applied Natural Language Processing Conference (ANLP-2000)*, pages 268–275, Seattle, WA, 2000.
113. A. Müller, J. Dörre, P. Gerstl, and R. Seiffert. The TaxGen Framework: Automating the Generation of a Taxonomy for a Large Document Collection. In *Proceedings of the 32^{nd} Hawaii International Conference on System Sciences (HICSS)*, page 2034, Maui, Hawaii, 1999. IEEE.
114. G. Navarro. *Approximate Text Searching*. PhD thesis, Universidad de Chile, 1998.
115. M.-J. Nederhof, G. Bouma, R. Koeling, and G. van Noord. Grammatical analysis in the OVIS spoken-dialogue system. In *Proceedings of the ACL/EACL Workshop on "Interactive Spoken Dialog Systems: Bringing Speech and NLP Together in Real Applications"*, Madrid, 1997.

116. R. Osdin, I. Ounis, and R. W. White. Using Hierarchical Clustering and Summarization Approaches for Web Retrieval: Glasgow at the TREC 2002 Interactive Track. In *Proceedings of the Eleventh Text Retrieval Conference (TREC-2002)*, NIST Special Publication 500-251, 2003.
117. S. M. Pahlevi and H. Kitagawa. Conveying Taxonomy Context for Topic-Focused Web Search. *Journal of the American Society for Information Science and Technology (JASIST)*, 56(2):173–188, 2005.
118. S. Parent, B. Mobasher, and S. Lytinen. An Adaptive Agent for Web Exploration Based on Concept Hierarchies. In *Proceedings of the 9^{th} International Conference on Human Computer Interaction (HCI)*, pages 903–907, New Orleans, 2001.
119. G. W. Paynter, I. H. Witten, S. J. Cunningham, and G. Buchanan. Scalable browsing for large collections: a case study. In *Proceedings of the 5^{th} ACM Conference on Digital Libraries*, pages 215–223, 2000.
120. J. Peckham. A new generation of spoken dialogue systems: results and lessons from the SUNDIAL project. In *Proceedings of the 3^{rd} European Conference on Speech Communication and Technology*, pages 33 – 40, Berlin, Germany, 1993.
121. J. M. Pierre. On the Automated Classification of Web Sites. *Linköping Electronic Articles in Computer and Information Science*, 6(1), 2001.
122. M. F. Porter. An Algorithm for Suffix Stripping. *Program*, 14(3):130–137, 1980.
123. Y. Qiu and H. P. Frei. Concept Based Query Expansion. In *Proceedings of the 16^{th} Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 160–169, Pittsburgh, Pennsylvania, 1993.
124. V. V. Raghavan and H. Sever. On the reuse of past optimal queries. In *Proceedings of the Eighteenth Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, Feedback Methods, pages 344–350, 1995.
125. L. F. Rau. Conceptual information extraction and retrieval from natural language input. In *Proceedings RIAO-88: Conference on User-Oriented, Content-Based, Text and Image Handling*, pages 424–437, Cambridge, MA, 1988.
126. P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl. GroupLens: An Open Architecture for Collaborative Filtering of Netnews. In *Proceedings of ACM CSCW'94 Conference on Computer-Supported Cooperative Work*, pages 175–186, 1994.
127. B. Rosario and M. Hearst. Classifying the Semantic Relations in Noun Compounds via a Domain-Specific Lexical Hierarchy. In *Proceedings of the 2001 Conference on Empirical Methods in Natural Language Processing (EMNLP 2001)*, pages 82–90, Pittsburgh, PA, 2001.
128. D. E. Rose and D. Levinson. Understanding User Goals in Web Search. In *Proceedings of the 13^{th} International World Wide Web Conference (WWW2004)*, pages 13–19, New York, 2004.
129. D. Roussinov, K. Tolle, M. Ramsay, and H. Chen. Interactive Internet Search through Automatic Clustering: an Empirical Study. In *Proceedings of the 22^{nd} Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 289–290, Berkeley, CA, 1999.
130. A. Rudnicky, E. Thayer, P. Constantinides, C. Tchou, R. Shern, K. Lenzo, W. Xu, and A. Oh. Creating natural dialogs in the Carnegie Mellon Com-

municator system. In *Proceedings of Eurospeech*, pages 1531–1534, Budapest, 1999.
131. I. Ruthven, A. Tombros, and J. M. Jose. A Study on the Use of Summaries and Summary-based Query Expansion for a Question-answering Task. In *Proceedings of the 23rd European Colloquium on Information Retrieval Research (ECIR'01)*, pages 41–53, Darmstadt, 2001.
132. A. Sahuguet and F. Aznavant. Looking at the Web through XML glasses. In *Proceedings of the 4th International Conference on Cooperative Information Systems (CoopIS'99)*, pages 148–159, Edinburgh, 1999.
133. G. Salton and M. J. McGill, editors. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Company, New York, 1983.
134. M. Sanderson and B. Croft. Deriving concept hierarchies from text. In *Proceedings of the 22nd Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 206–213, Berkeley, CA, 1999.
135. B. Santorini. Part-of-speech tagging guidelines for the Penn Treebank Project. Technical report MS-CIS-90-47, Department of Computer and Information Science, University of Pennsylvania, 1990.
136. J. Savoy and J. Picard. Report on the TREC-8 Experiment: Searching on the Web and in Distributed Collections. In *Proceedings of the Eighth Text Retrieval Conference (TREC-8)*, pages 229–240, NIST Special Publication 500-246, 1999.
137. N. Shadbolt, N. Gibbins, H. Glaser, S. Harris, and M. C. Schraefel. CS AKTive Space, or How we Learned to Stop Worrying and Love the Semantic Web. *IEEE Intelligent Systems*, 19(3):72–79, May/June 2004.
138. C. Silverstein, M. Henzinger, and H. Marais. Analysis of a Very Large AltaVista Query Log. Digital SRC Technical Note 1998-014, 1998.
139. A. Singhal and M. Kaszkiel. A Case Study in Web Search using TREC Algorithms. In *Proceedings of the Tenth International World Wide Web Conference (WWW10)*, pages 708–716, Hong Kong, 2001.
140. A. F. Smeaton. Using NLP or NLP Resources for Information Retrieval Tasks. In T. Strzalkowski, editor, *Natural Language Information Retrieval*, pages 99–111. Kluwer Academic Publishers, 1997.
141. D. Smith and M. Lopez. Information extraction for semi-structured documents. In *Proceedings of the "Workshop on Management of Semi-Structured Data"*, pages 60–66, Tucson, Arizona, 1997.
142. D. Smith and M. Lopez. Information finding and filtering for collections of semi-structured documents. In *Proceedings of INFORSID XV*, pages 353–367, Toulouse, 1997.
143. S. Soderland. Learning Information Extraction Rules for Semi-Structured and Free Text. *Machine Learning*, 34(1-3):233–272, 1999.
144. R. Song, H. Liu, J.-R. Wen, and W.-Y. Ma. Learning Important Models for Web Page Blocks based on Layout and Content Analysis. *SIGKDD Explorations Newsletter*, 6(2):14–23, 2004.
145. K. Spärck Jones. Document Retrieval: Shallow Data, Deep Theories; Historical Reflections, Potential Directions. In F. Sebastiani, editor, *Proceedings of the 25th European Colloquium on Information Retrieval Research (ECIR'03)*, Lecture Notes in Computer Science 2633, pages 1–11, Pisa, 2003. Springer Verlag.
146. T. Strzalkowski, L. Guthrie, J. Karlgren, J. Leistensnider, F. Lin, J. Perez-Carballo, T. Straszheim, J. Wang, and J. Wilding. Natural Language Infor-

mation Retrieval: TREC-5 Report. In *Proceedings of the Fifth Text Retrieval Conference (TREC-5)*, pages 291–314, NIST Special Publication 500-238, 1997.
147. T. Strzalkowski, J. Perez-Carballo, J. Karlgren, A. Hulth, P. Tapanainen, and T. Lahtinen. Natural Language Information Retrieval: TREC-8 Report. In *Proceedings of the Eighth Text Retrieval Conference (TREC-8)*, pages 381–390, NIST Special Publication 500-246, 1999.
148. H. Stuckenschmidt, A. de Waard, R. Bhogal, C. Fluit, A. Kampman, J. van Buel, E. van Mulligen, J. Broekstra, I. Crowlesmith, F. van Harmelen, and T. Scerri. A Topic-Based Browser for Large Online Resources. In E. Motta, N. Shadbolt, A. Stutt, and N. Gibbins, editors, *Proceedings of Engineering Knowledge in the Age of the Semantic Web, 14^{th} International Conference, EKAW 2004*, Lecture Notes in Computer Science 3257, pages 433–448. Springer Verlag, 2004.
149. Süddeutsche Zeitung Magazin, December 2000. Number 52.
150. K. Summers. *Automatic Discovery of Logical Document Structure*. PhD thesis, Cornell University, 1998.
151. R. F. E. Sutcliffe and K. White. Searching via keywords or concept hierarchies - which is better? In *Proceedings of the 3^{rd} International Conference on Language Resources and Evaluation*, pages 2103–2106, Las Palmas de Gran Canaria, Spain, 2002.
152. K. Taghva, A. Condit, and J. Borsack. Autotag: A Tool for Creating Structured Document Collections from Printed Material. Technical Report TR 94-11, Information Science Research Institute, University of Nevada, 1994.
153. P. D. Turney. Extraction of Keyphrases from Text. Technical Report ERB-1057, National Research Council of Canada, Institute for Information Technology, 1999.
154. P. D. Turney. Learning Algorithms for Keyphrase Extraction. *Information Retrieval*, 2(4):303–336, 2000.
155. C. J. van Rijsbergen. *Information Retrieval*. Butterworths, 1979.
156. E. Voorhees and D. Harman. Overview of TREC 2001. In *Proceedings of the Tenth Text Retrieval Conference (TREC-2001)*, pages 1–15, NIST Special Publication 500-250, 2002.
157. N. Wacholder, D. K. Evans, and J. L. Klavans. Automatic identification and organization of index terms for interactive browsing. In *Proceedings of the ACM/IEEE Joint Conference on Digital Libraries, JCDL 2001*, pages 126–134. ACM, 2001.
158. W. Wahlster, editor. *Verbmobil: Foundations of Speech-to-Speech Translation*. Springer Verlag, Berlin, 2000.
159. M. Walker, L. Hirschman, and J. Aberdeen. Evaluation for DARPA Communicator Spoken Dialogue Systems. In *Proceedings of the 2^{nd} International Conference on Language Resources and Evaluation*, pages 735–741, Athens, Greece, 2000.
160. M. Walker, C. Kamm, and D. Litman. Towards developing general models of usability with PARADISE. *Natural Language Engineering*, 6(3):363–377, 2000.
161. N. Webb, A. De Roeck, U. Kruschwitz, P. Scott, S. Steel, and R. Turner. Evaluating a Natural Language Dialogue System: Results and Experiences. In *Proceedings of the Workshop "From Spoken Dialogue to Full Natural Interactive Dialogue - Theory, Empirical Analysis and Evaluation" (at the 2^{nd}*

International Conference on Language Resources and Evaluation LREC2000), pages 22–26, Athens, Greece, 2000.

162. X. Wei and A. Rudnicky. Task-based dialog management using an agenda. In *ANLP/NAACL 2000 Workshop on Conversational Systems*, pages 42–47, Seattle, 2000.
163. R. Weiss, B. Velez, M. A. Sheldon, C. Nemprempre, P. Szilagyi, A. Duda, and D. K. Gifford. HyPursuit: A Hierarchical Network Search Engine that Exploits Content-Link Hypertext Clustering. In *Proceedings of the Seventh ACM Conference on Hypertext*, pages 180–193, Washington DC, 1996.
164. R. W. White, J. M. Jose, and I. Ruthven. Comparing Explicit and Implicit Feedback Techniques for Web Retrieval: TREC-10 Interactive Track Report. In *Proceedings of the Tenth Text Retrieval Conference (TREC-2001)*, pages 534–538, NIST Special Publication 500-250, 2002.
165. R. W. White, I. Ruthven, and J. M. Jose. The Use of Implicit Evidence for Relevance Feedback in Web Retrieval. In F. Crestani, M. Girolami, and C. J. van Rijsbergen, editors, *Proceedings of the 24^{th} European Colloquium on Information Retrieval Research (ECIR'02)*, Lecture Notes in Computer Science 2291, pages 93–109. Springer Verlag, 2002.
166. D. Widdows, S. Cederberg, and B. Dorow. Visualisation Techniques for Analysing Meaning. In *Text, Speech, and Dialogue. Fifth International Conference (TSD2002)*, pages 107–114, 2002.
167. D. Widdows and B. Dorow. A Graph Model for Unsupervised Lexical Acquisition and Automatic Word-Sense Disambiguation. In *Proceedings of the 19^{th} Conference on Computational Linguistics (COLING)*, pages 1093–1099, Taipei, Taiwan, 2002.
168. R. Yangarber, R. Grishman, P. Tapanainen, and S. Huttunen. Automatic Acquisition of Domain Knowledge for Information Extraction. In *Proceedings of the 18^{th} Conference on Computational Linguistics (COLING)*, pages 940–946, Saarbrücken, 2000.
169. O. Zamir and O. Etzioni. Grouper: A Dynamic Clustering Interface to Web Search Results. In *Proceedings of the Eighth International World Wide Web Conference (WWW8)*, pages 1361–1374, Toronto, 1999.
170. H.-J. Zeng, Q.-C. He, Z. Chen, W.-Y. Ma, and J. Ma. Learning to Cluster Web Search Results. In *Proceedings of the 27^{th} Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 210–217, Sheffield, 2004.
171. C. Zhai. Fast Statistical Parsing of Noun Phrases for Document Indexing. In *Proceedings of the 5^{th} Conference on Applied Natural Language Processing*, pages 312–319, Washington DC, 1997.
172. R. Y. Zhang, L. V. S. Lakshmanan, and R. H. Zamar. Extracting Relational Data from HTML Repositories. *SIGKDD Explorations Newsletter*, 6(2):5–13, 2004.
173. V. Zue. Toward Systems that Understand Spoken Language. *IEEE Expert Magazine*, 9(1):51–59, February 1994.
174. V. Zue, J. Glass, D. Goodine, H. Leung, M. Phillips, J. Polifroni, and S. Seneff. The VOYAGER Speech Understanding System: Preliminary Development and Evaluation. In *Proceedings of IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 73–76, 1990.

Index