Concept-Based Retrieval using Controlled Natural Language

Osmar R. Zaïane[1] Andrew Fall[1] Stephen Rochefort[1]
Veronica Dahl[1] Paul Tarau[2]

[1] School of Computing Science
Simon Fraser University
Burnaby, B.C., Canada V5A 1S6
{zaiane, fall, srochefo, veronica}@cs.sfu.ca

[2] Department of Computing Science
University of Moncton
Moncton, Canada E1A 3E9
tarau@info.umoncton.ca

Abstract

We present a method for retrieving concepts from web search queries and from candidate documents on the web, to help determine which of these documents are semantically (rather than simply key-word wise) related to the query. Our method combines *hypothetical reasoning*, which we use both for natural language analysis and for concept extraction, and domain-oriented taxonomies of concepts to guide the system's reasoning.

1 Introduction

Realistic natural language analysis, whether for web or traditional applications, cannot make abstraction of semantics and pragmatics, any more than programming languages can fully make abstraction of their run-time environments. Computer-based discourse understanding (as its human counterpart) is basically a form of model-building. It involves constant constraint-solving to keep, at a given time, only a manageable subset of (intended) models. The task is harder, but similar to that of compilers for programming languages.

The experience of being drowned with documents as a result of a keyword-based web search is all too common. For instance, let us consider what happens when we ask two of the existing popular search tools for "clear cuts near water", a forestry related query: *Lycos* [19] responds with "Complete poetical works from William Wordsworth", among a list of other equally wrong associations. *Excite*, which succeeds with more specific queries such as "Clear cuts near river", also produces nonsense replies to the query "clear cuts near water", such as "Plumbing frequently asked questions". It is important to understand why this happens. The "Plumbing frequently asked questions", for example, simply contains occurrences of "water", "near", "cuts" and "clear" and thus, is indexed under these words in the inverted index. Another document about cleaning toys might have contained the same words with high frequency and would have appeared in the response list.

Our approach, in contrast, can use a taxonomy of forestry-related concepts which allows it to specialize or generalize given concepts (e.g. going from "water" to "lakes" or vice versa), and thus is able to use the contextual information provided by forestry domains in order to avoid nonsensical answers. Therefore, a document like "Plumbing frequently asked questions" would not be indexed with the words it contains, but with its semantic content.

We also use hypothetical reasoning, in two ways:

1. in the form of Assumption Grammars (our natural language processing tool, based on logic programming), and

2. in the form of an assumption-handling mechanism included in BinProlog [26], which we use both to help match requested concepts to those found in documents, and to dynamically revise concept hierarchies.

In this paper we focus on the specific point of how to use natural language for retrieving or matching concepts. Other components of our system are described only in enough detail to understand the general picture. We refer to a companion paper [27] for a detailed description of our domain-oriented taxonomies.

2 Today's Search Engines will Kick the Bucket

How often have you been unsuccessful in finding what you wanted in an on-line search because the words you used failed to match words in the material that you needed? The frustration is greater when an existing document, directly relevant to your quest, does not appear in the list presented by the search engine as a response to your query. When presenting idiomatic phrases, like "kick the bucket", search engines usually come-up with hilarious answers. The causes are many. The important factors that make a search engine effective are the way documents are analyzed and indexed, the techniques provided to the user to submit queries, and the ranking method used to sort the list of documents in the response list.

The techniques used to index on-line documents are simple and crude. Because of the dynamic nature and the size of the WWW, speed and efficiency of these techniques are primordial. Some search engines index documents based on the words in their titles or headers (i.e. section titles). This method is fast, generates small indexes, but is inaccurate since it may miss pertinent topics from the main body of the document. Other techniques simply take all the words from a document and index the document based on all these words, creating an inverted index. These techniques produce very big indices, from 10 to 40% of the document original size, and are more accurate, but still generate a lot of *noise*[1] in query answers. Some variations of the "index all words" technique tend to reduce the number of words by eliminating "empty" words like articles (i.e. the, a, etc.) or common verbs (i.e. is, do, have, etc.), or aggregating words from the same canonical form (e.g.: clearing, cleared, clears, clear). This reduces the size of the inverted index and thus accelerates the search. The aggregation of words from the same canonical form reduces *silence*[2]. Some search engines attempt to reduce the silence effect (without adding noise) by aggregating some common synonyms.

The means provided to the user to submit queries is usually standard keyword based. Some advanced interfaces allow conjunctions of keywords, combinations of disjunctions of keywords, and even negations. The boolean combination of keywords is evaluated and matched with the words in the inverted index, built in advance, to retrieve the identification codes of the documents containing these keywords. Some search engines, like Alta Vista[3], allow entering exact phrases instead of simple keywords. This generally makes the inverted index larger and more complex.

Finally, the ranking methods applied to sort the document list presented to the user, is used to order the documents by relevance to the query. The rank of a document is usually based on the conformity of

[1]Noise is an irrelevant document which appears in the answer list.

[2]Silence happens when an existing document, relevant to the user's subject of interest, does not show up in the response list given by the search engine.

[3]Alta Vista is available at: http://www.altavista.digital.com/

the keywords of the query and the document's keywords, as well as the occurrence of the keywords in the document. Some ranking formulas may also include in the ranking the relative position of keywords in the document.

Today's search engines use brute force to index on-line documents for the sake of simplicity. This was acceptable for a relatively small global information network or a local document database. However, the bigger the WWW becomes, the less satisfactory the answers from today's search engines will become. Changing the ranking formulas will not be sufficient to balance the reduction of noise and silence. Some search engines added new features, like date of document, Internet domain of origin (location), and even format of the document, to narrow the search. These are intermediate solutions that temporarily ease the user's frustration.

An ideal document retrieval system should allow natural language (or pseudo natural language) querying, and should index documents based on the concepts present in them. Documents containing idioms like "kick the bucket", "bite the dust" or "meet its maker", should be indexed by the concept "death". The use of concept classification could allow the application of relationships like *parent-child* or *sibling* to link the concept "death" from a query to concepts like "funeral", "obituary" or "suicide" present in documents. The use of concept hierarchies could also allow the introduction of qualifiers such as "like", "close-to", "related-to", etc., in the query language to help the user refine the request.

Concept-based retrieval systems attempt to reach beyond the standard keyword approach of simply counting the words from the request that occur in a document. The conceptual indexing system we present in the next section attempts to extract pertinent subjects from documents to categorize these documents by concepts, and uses knowledge of concepts and their interrelationships to find correspondences between the concepts in the request and those that occur in the documents.

3 Our General Framework

The World Wide Web, as the universal electronic library it has become, lacks one useful thing: a good card catalogue system.

There are software tools, such as the *World Wide Web Worm* [20], *Lycos*[4], *Excite*[5] and others, which can find articles containing user-supplied strings of characters; it is usually possible to locate the e-mail address of any author. However, a user attempting to do a subject search of the available documents will find it very difficult. Most use spider-based indexing techniques like RBSE [9] which systematically crawl the web to access as many on-line documents as possible. These techniques, in general, because they emphasize on speed, not only flood the network and servers but also lose the structure and context of documents. AltaVista, which claims to have the fastest indexing scheme, spends six weeks to go over all accessible documents on the WWW. Other indexing solutions, like ALIWEB [17] or Harvest [3], behave well on the network but still struggle with the difficulty to isolate information with relevant context.

In physical libraries, skilled catalogers handle the problem of trying to classify the subjects of the books they receive from publishers, working from standard lists of subject headings. On the Internet however, users supply documents themselves. The rapidity with which the available information grows, changes and is made obsolete, makes professional cataloging of documents infeasible. The document providers are also

[4]Lycos is available at http://www.lycos.com
[5]Excite is available at http://www.excite.com

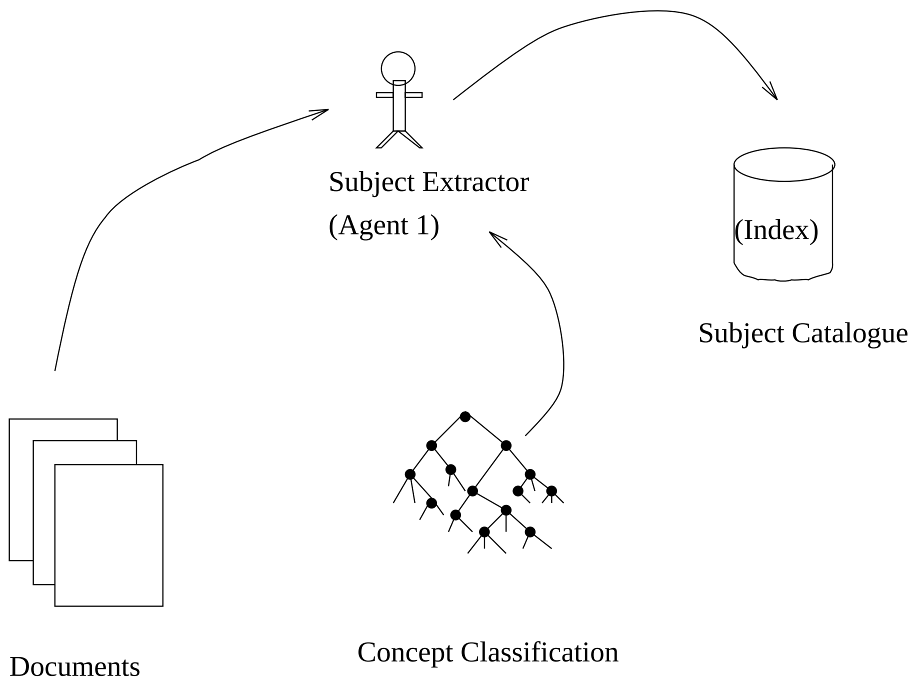


Figure 1: Subject Extractor Expert indexing selected resources.

notoriously lax at taking the time and effort to supply subject classifications for their information, let alone complete and standardized ones.

It seems that what we need is a computational agent which will at least approximately classify documents according to their subjects, as a first step toward a subject index of the library that is the Web. Such an agent, an "automated librarian", would read and assign documents to appropriate categories.

Many search engines (programs and services designed to help users search for documents of interest to them) have been implemented for the WWW. Netcreations[6], which has a URL announcement service, lists 402[7] different search engines and directories on the Web. We know of none that does anything but textual pattern-matching on the titles and headers of documents or the entire text of documents, except Excite which claims to use some concept classifications.

As a first step toward a concept index of the library that is the Web, a computational agent which, at least, approximately classifies documents according to their subjects, is needed. We call the agent *Subject Extractor*. The Subject Extractor extracts relevant topics from a given document.

We have already seen in the Introduction how we can avoid nonsensical answers such as those for queries like "clear cuts near river", by using a taxonomy of forestry-related concepts.

Using Assumption Grammars and Concept Classifications, the *Subject Extractor*, as shown in Figure 1, is used to educe relevant topics from given documents. The topics are then stored with the document identification in a subject catalogue for future matches with document retrieval requests (Figure 3). The subject catalogue uses the same concept classification to order the index in a multi-layered database. Figure 2 shows how the same *Subject Extractor* can be used to educe topics of interest from a user query written in natural language or controlled English.

[6]http://www.netcreations.com
[7]The list is accessible at http://www.netcreations.com/postmaster/thelist.html

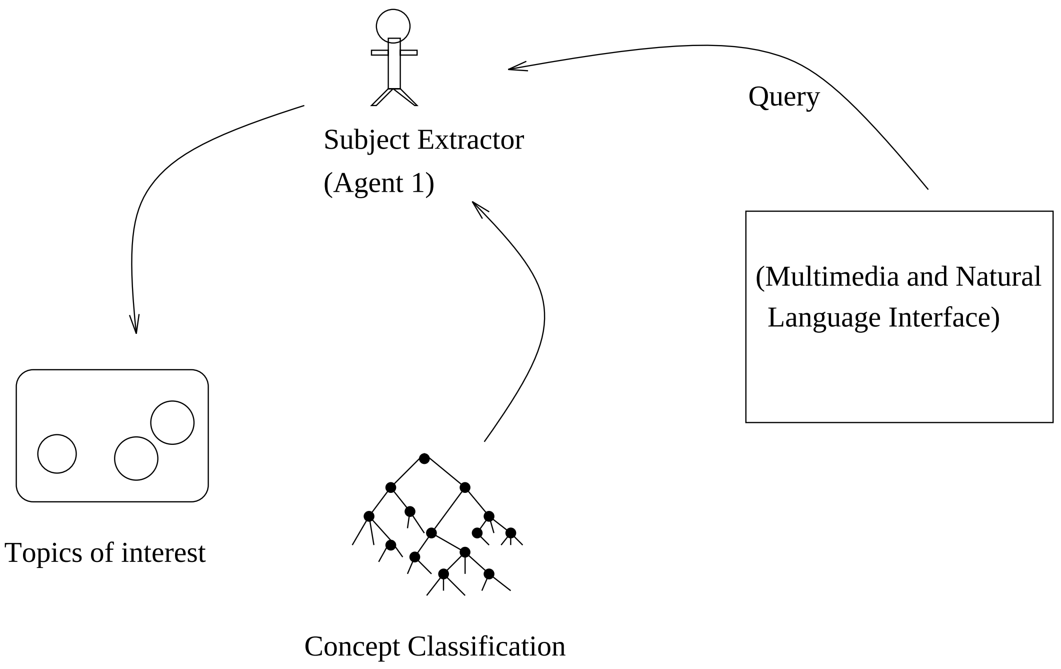


Figure 2: Subject Extractor Expert extracts topics from the pseudo natural language query.

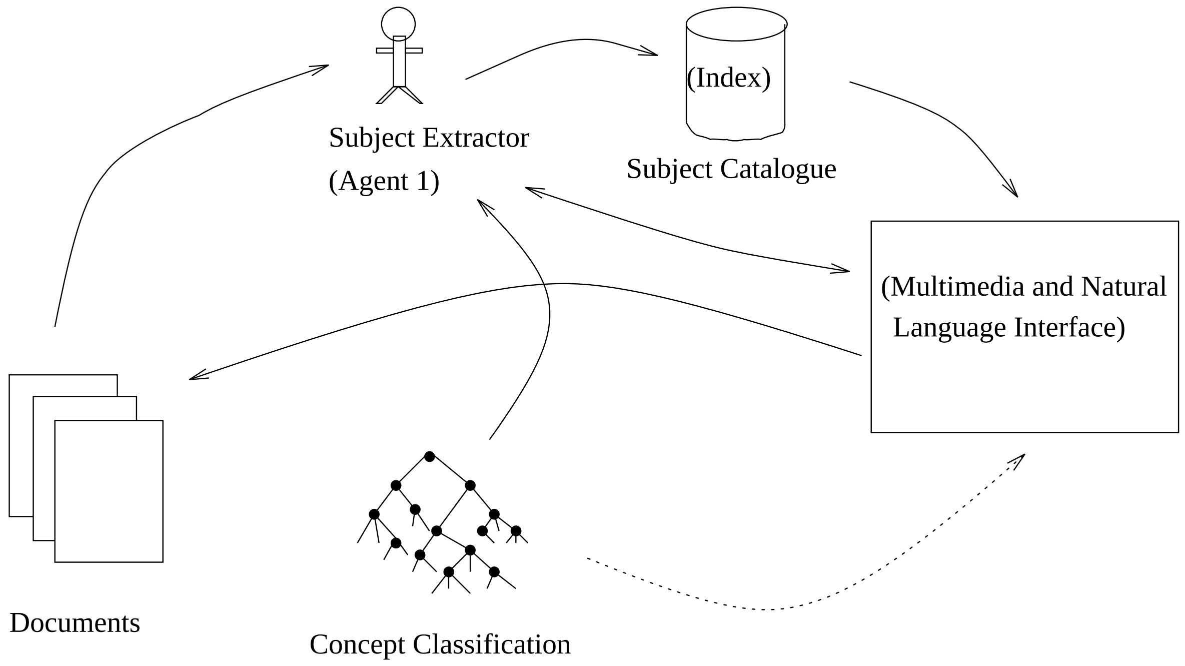


Figure 3: Pseudo Natural Language Interface using the subject catalogue and concept classification to retrieve documents.

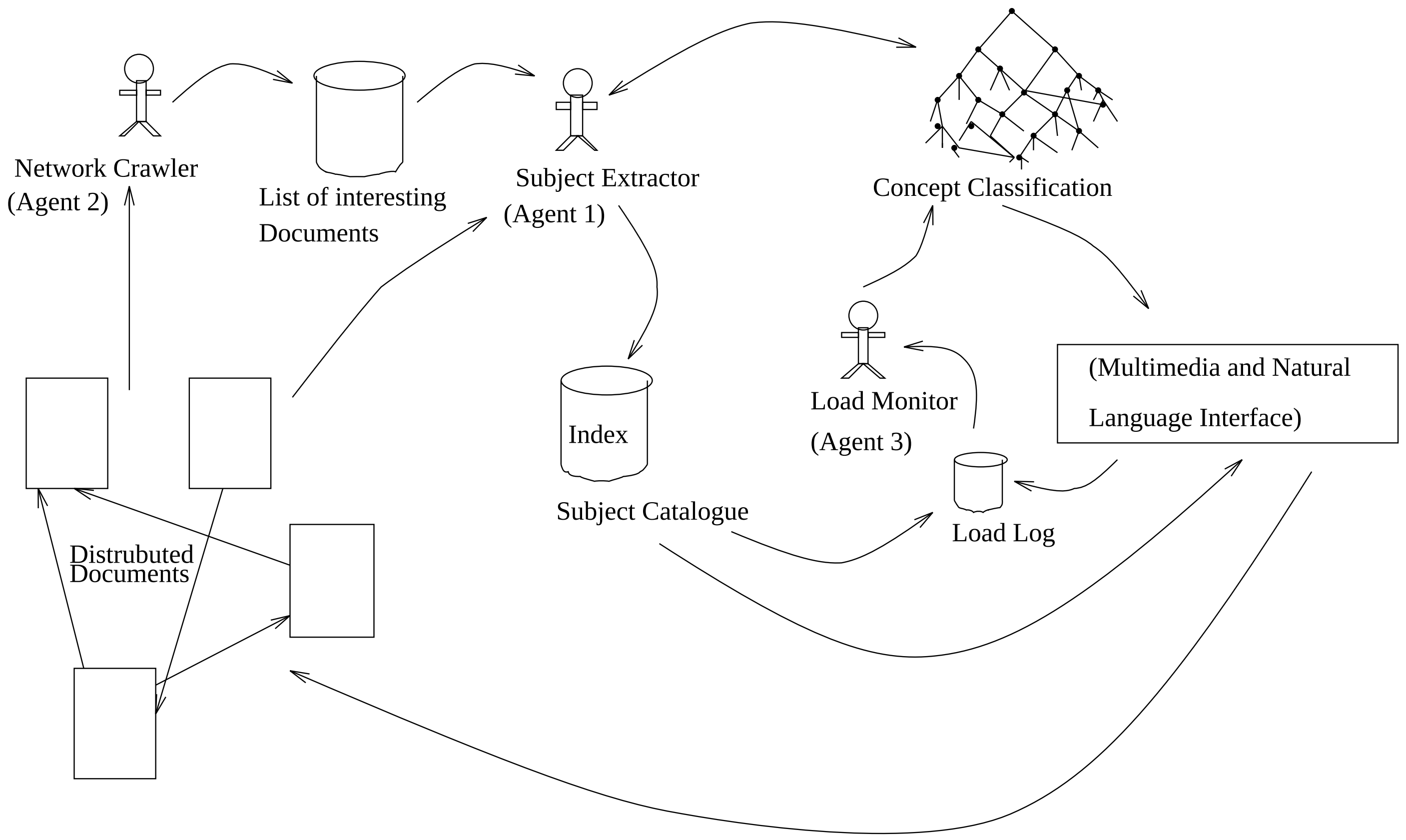


Figure 4: The Concept-Based Retrieval System applied to the global information Network with an intelligent concept classification updating agent.

With the Subject Extractor and the Subject catalogue and using the Artificial Intelligence tools described in this article, we propose a WWW agent for searching, indexing and organizing information relative to a given industrial context (Figure 4). We call this second agent *Network Crawler* which is a soft robot (also called wanderer or spider). The Network Crawler works in conjunction with the Subject Extractor to index and organize documents. It starts from a given web page, gives its content to the subject extractor for processing, then, like any web spider and following the standard for robot exclusion[8], recursively follows all the hyper-links in the document to load other web pages. The search of documents can be restricted to a given directory, a given web site, or a network domain.

In Figure 4, the *Load Monitor* is an agent which observes the usage of the Concept Classification and keeps a statistical record of the employment of paths in the concept hierarchy in order to optimize the use of the concept classification by eliminating paths that have been used less often (i.e. by generalizing sibling concepts) or by reorganizing the overused paths in more a efficient way (i.e. by specializing concepts and adding new subsumed concepts). Figure 4 represents the complete concept-based retrieval system for a global information network.

4 Description of Some of Our Tools

All the tools described in this section will be used by the Subject Extractor.

4.1 Hypothetical Reasoning Tools

Intuitionistic `*/1` adds temporarily a clause usable in later proofs. Such a clause can be used an indefinite number of times, like asserted clauses, except that it vanishes on backtracking. *Linear* `+/1` adds temporarily a clause usable *at most once* in later proofs. Being usable *at most once*(i.e., *weakening* is allowed), together with implicit scoping over the current continuation, distinguishes *affine* linear logic from Girard's original framework where linear implications should be used *exactly* once. This assumption also vanishes on backtracking. Both types of assumptions have a scoped version which will not concern us here.

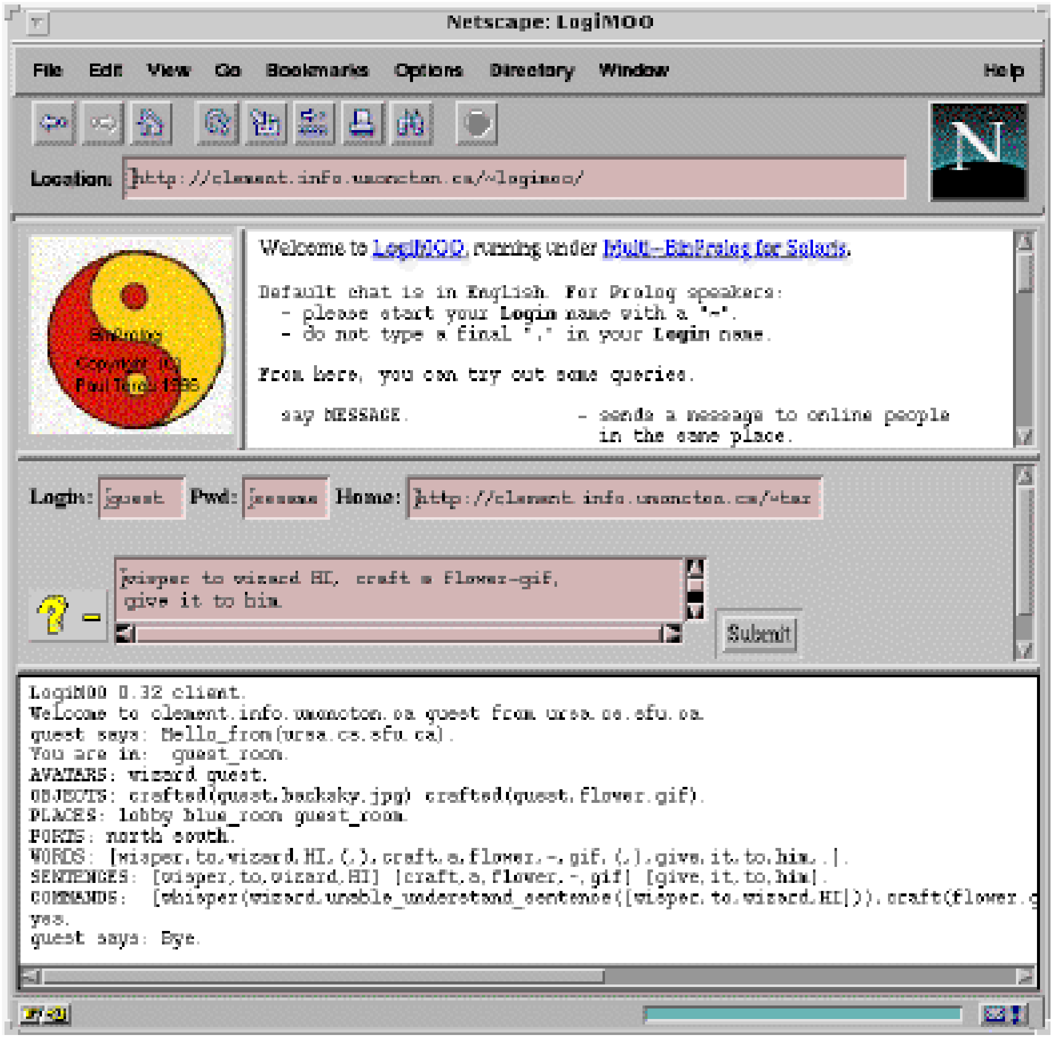
We can see the `+/1` and `*/1` built-ins as linear affine and respectively intuitionistic implication scoped over the current AND-continuation, i.e. having their assumptions available in future computations on the *same* resolution branch.

Notice that to disallow *weakening*, we can simply specify that the set of left over assumptions should be empty (this is easy through negation as failure, and can be made invisible to the user to maintain declarativeness).

4.2 Assumption Grammars

Assumption Grammars[5] can be used for representing natural language so that it becomes executable (i.e., so that automatic inferences can be made from our description which result in automatic translations between natural language and meaning representation). Such a meaning representation is then used to consult our system and extract answers to our queries. We have shown how to solve several crucial language processing problems (i.e. coordination, anaphora, free word order) through Assumption Grammars [7]. A first Assumption Grammar prototype for English has already been successfully produced for another web

[8]Non enforced set of rules that if followed would protect web servers from unwanted accesses by robots. Accessible at http://info.webcrawler.com/mak/projects/robots/norobots.html


Netscape: LogiMOO
File Edit View Go Bookmarks Options Directory Window Help
Location:
BinProlog Copyright (c) Paul Tarau 1996
Login:
Pwd:
Home:
Submit

http://clement.info.umoncton.ca/~logimoo/

Welcome to LogiMOO, running under Multi-BinProlog for Solaris.

```
Default chat is in English. For Prolog speakers:
  - please start your Login name with a '-'.
  - do not type a final '.' in your Login name.

From here, you can try out some queries.

  say MESSAGE.                  - sends a message to online people
                                  in the same place.
```

Login: guest Pwd: ****** Home: http://clement.info.umoncton.ca/~tar

```
wisper to wizard HI, craft a flower-gif,
give it to him
```

```
LogiMOO 0.32 client.
Welcome to clement.info.umoncton.ca guest from ursa.cs.sfu.ca
guest says: Hello_from(ursa.cs.sfu.ca).
You are in:  guest_room.
AVATARS: wizard guest.
OBJECTS: crafted(guest,backsky.jpg) crafted(guest,flower.gif).
PLACES: lobby blue_room guest_room.
PORTS: north south.
WORDS: [wisper,to,wizard,HI,(,),craft,a,flower,-,gif,(,),give,it,to,him,.|.
SENTENCES: [wisper,to,wizard,HI] [craft,a,flower,-,gif] [give,it,to,him].
COMMANDS:  [whisper(wizard,unable_understand_sentence([wisper,to,wizard,HI])),craft(flower.g
yes.
guest says: Bye.
```

Figure 5: LogiMOO User Interface using Controlled English.

application [25], LogiMOO. Figure 5 provides a view of the LogiMOO interface. This prototype is easily adaptable to other domains by adjusting the lexicon corresponding to nouns, verbs and adjectives. We maintain this approach in the present work so that adapting the natural language processor to different domains will not be too complex a task.

Assumption Grammars are basically logic grammars, like Definite Clause Grammars, except that they are augmented with:

1. linear and intuitionistic implications scoped over the current continuation
2. hidden multiple accumulators, useful in particular to make the input and output strings invisible.

Hidden multiple accumulators are important for efficiency and for other uses than the one we exploit in this paper, but for our purposes here, we can disregard them.

4.3 Domain-Oriented Lexicons - Incomplete Types

The mapping words into concepts, and (through the words found in candidate documents) of documents into concepts proceeds largely through domain-oriented lexicons based on hierarchical representations of the concept or type associated to each "content" word (namely, nouns, verbs and adjectives). For each domain, experts will be consulted to arrive at domain-oriented mappings between words and concepts. Note that there may still be ambiguities within the domain chosen- these can then be further disambiguated with respect to the sentence context.

Briefly, what we do is associate an incomplete type [4] to each noun, verb and adjective in the lexical definition. Incomplete types are convenient representations of taxonomic representations, which allow us to reduce the length of the proofs needed to deduce some set-inclusion relationships among concepts from a hierarchy of concepts, to provide intensional replies, to perform quick semantic agreement verifications on natural language queries, and to achieve partial execution of some queries.

By having access through the lexicon to the concept(s) associated to a given word, the parser can not only discard word meanings that would be acceptable in general but not in the particular domain described, but also choose among alternative meanings still possible within the domain given by type compatibility constraints embedded in the parser.

For instance, the lexical entry for: "logging" in a domain where trees and computers are both relevant could specify that if the direct object required by the very is "trees", the concept referred to is that of cutting, not that of "exiting the computer session". These constraints are checked by the analyzer simply by matching, through unification, that the expected object of the verb coincides in type with that obtained by the actual object as found in the lexical entry for that object (or rather, in the lexical entry for the object's head noun).

4.4 Concept Classification Tools

For the concept classification component, in the computer generated phase, we use methodologies for the efficient management of hierarchies. The fundamental basis for the concept classification in our system is a *partial order* of concepts: a pair $(\Sigma,\leq)$, were Σ is a set of concepts and $\leq$ is a reflexive, anti-symmetric and transitive relation, called *subsumption*, among those concepts [8]. In a graphical viewpoint this is equivalent to a directed acyclic graph, where more general concepts appear above more specialized concepts.

An example concept classification is shown in Figure 6. Note that the graph is not limited to a tree form - some nodes may have multiple parents. For example, the *Data Mining* concept is subsumed by the concepts for *Statistics*, *Machine Learning* and *Database Systems.*

The primary operations on concept hierarchies include testing the subsumption relation, and computing *greatest lower bounds* and *least upper bounds.* For example, in Figure 6, the relation *Data Mining* $\leq$ *Science* holds, but not *Phonetics* $\leq$ *Science.*

The greatest lower bound, $\sqcap$, of a set of concepts A is the most general concept (if one exists) that is subsumed by all the concepts in A. For example, *Computer Science* $\sqcap$ *Linguistics* = *Natural Language Processing.* A greatest lower bound may not exist for a set of concepts, or there may be more than one maximal lower bound. Our system for managing hierarchies deals with all these situations. The least upper bound, $\sqcup$, of a set of concepts A is the dual of a greatest lower bound. It is the most specific concept (if one exists) that subsumes all the concepts in A. For example *Data Mining* $\sqcup$ *Physics* = *Science.*

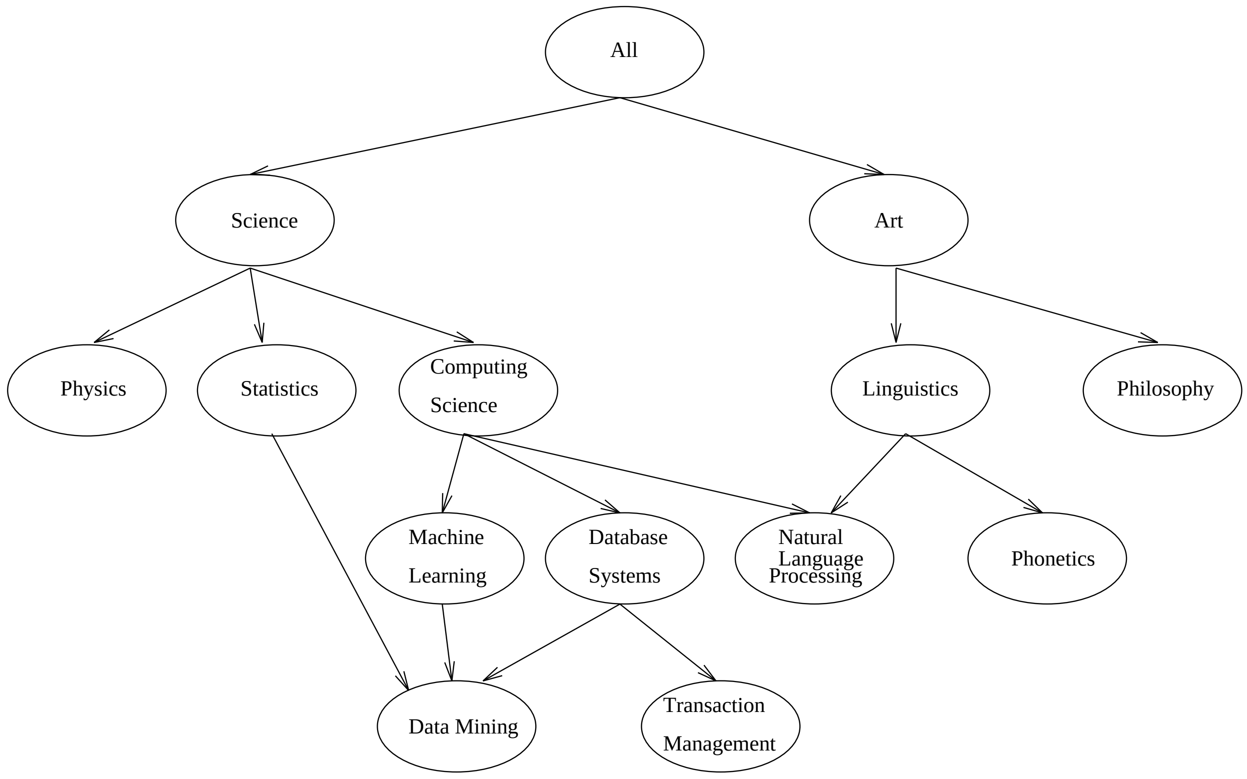


Figure 6: An example of a Concept Classification.

If greatest lower bounds and least upper bounds exist for all pairs of concepts, then a partial order is called a *lattice* [8]. Although some systems require hierarchies to be lattices, we feel that the more general structure of a partial order permits more intuitive and less restrictive specification of the concept classification.

There are a number of important issues regarding the construction, representation and use of such hierarchies. Obtaining relevant concept classifications is a non-trivial task. In the current system, we hand-generate the concept classification using domain specific knowledge of the target application area.

As concept classifications grow in size, the need for efficient representations or *encodings* of hierarchies becomes important [12]. Hierarchical encoding techniques have been developed for a variety of tasks, from knowledge representation [10], natural language processing [21] and logic programming [1], to use in databases [23] and operating systems [18]. These methodologies encode a hierarchy already given (either hand-generated or automatically generated) by associating with each element in the hierarchy a carefully constructed code which allows for efficient subsequent consultation. This is obtained by taxonomic encoding techniques which reduce expensive hierarchical operations to inexpensive set operations [11]. We have developed algorithms for the efficient encoding and management of concept classifications in dynamic environments [13].

4.5 Multi-Layered Databases and Concept Hierarchies

We use a different approach, called a Multiple Layered DataBase (MLDB) to represent our indices. This approach uses our concept hierarchies to classify the inverted indices, which in turn would facilitate information discovery in global information systems [28]. A multiple layered database (MLDB) is a database composed of several layers of information, with the lowest layer (i.e., *layer-0*) corresponding to the primitive information

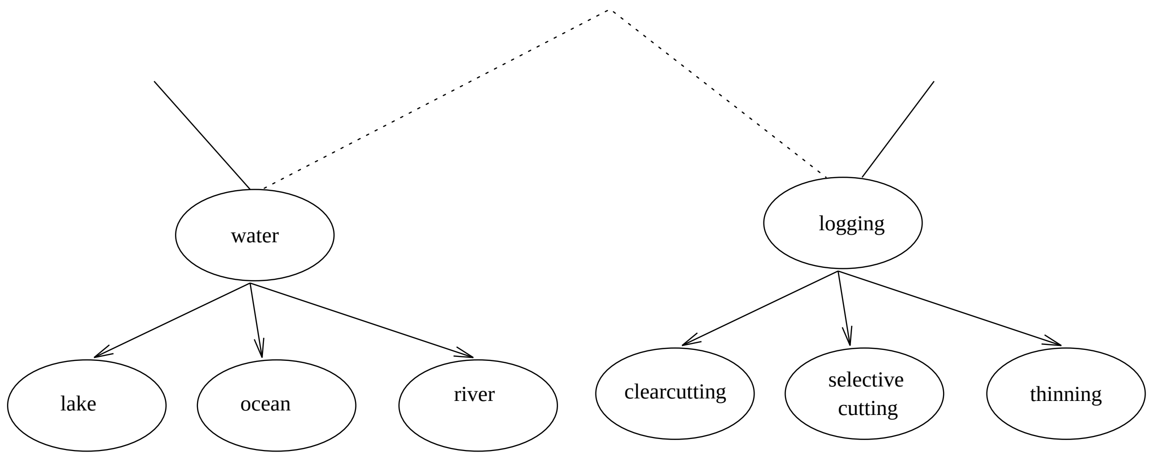


Figure 7: Two sub-trees from a concept classification.

Table 1: Layer 0

Clearcutting	D1, D2, D3, D11, D15
Lake	D2, D6, D8
Ocean	D5, D14
River	D7, D9, D10, D11
Selective cutting	D5, D9, D13, D14
Thinning	D6, D7, D12

Table 2: Layer 1

Logging	D1, D2, D3, D4, D5, D6, D7, D9, D11, D12, D13, D14, D15
Water	D2, D5, D6, D7, D8, D9, D10, D11, D14

Figure 8: Example of inverted index in a MLDB structure.

stored in the global information base and the higher ones (i.e., *layer-1* and above) storing generalized information extracted from the lower layers. Every *Layer* i generalizes *Layer* $i-1$. In other words, concepts present in *Layer* $i-1$ are subsumed by concepts present in *Layer* i. The proposal is based on the previous studies on *multiple layered databases* [24, 15] and *data mining* [22, 14, 16].

Building our indices in a multiple-layered database is simple. The inverted index constitutes Layer 0. In our case, the entries in the inverted index are not words from the documents, but are the subjects extracted. Each entry in the Layer 0 is a subject (i.e. concept) and the set of documents related to it. The subsequent layers are built by going up in the concept hierarchy for each concept, and uniting the sets of documents related to all subsumed concepts. In other words, the entry in *Layer* i is associated with all its descendents in *Layer* $i-1$. Notice that there may be additional documents at the higher level. In the example using the concept hierarchy shown in Figure 6, some documents can be indexed under "Linguistics" but not "Phonetics" or "Natural Language Processing". This can continue to Layer n (where n is the depth of the concept hierarchy), or can be stopped at an arbitrary lower level of the concept hierarchy. The set of documents related to a concept is a subset of the documents related to a subsuming concept. Note that any sub-order of the concept hierarchy can be used to create a specialized index.

Let's take an example: assuming we have 15 documents about logging practices and locations, and the two sub-trees from the concept hierarchy presented in Figure 7, table 1 in Figure 8 can be the inverted index containing the concepts extracted and the identification of the documents from which they were extracted. The concept "thinning", for instance, is in documents D6, D7, and D12. Table 1 in Figure 8 is considered Layer 0. "Water" subsumes "lake", "ocean" and "river", therefore, when building Layer 1, the set of documents associated with "water" is the union of all sets of documents containing concepts from Layer 0 subsumed by "water". Table 2 in Figure 8 shows Layer 1. Note that although D4 does not appear in table 1, it is indexed under "logging". D4 contains the concept "logging" but none of "clearcutting", "selective cutting" or "thinning" concepts. D8 is indexed under "water" but not "logging". Apparently, the topic logging or logging practices were not present in the document.

Inverted indices are used by search engines to immediately find documents that contain a given keyword. The list of keywords is sorted, and by using a B-tree structure, the number of I/O accesses is reduced significantly. With the multi-layered database approach, this still holds true. Each layer is an inverted index generalizing the previous layer, and specializing the next one. The main advantage of the multi-layered database architecture is its progressive search capability. A user can start by using high level concepts, then progressively narrow the search by drilling down in the concept hierarchy. For example, using Layer 1 and 0 in tables from Figure 8, and submitting the query *"logging near water"*, we would get the answer D2, D5, D6, D7, D9, D11, and D14, which is the intersection between the "logging" and the "water" entries in Layer 1. By specializing the concept "water" to "river", the query becomes *"logging near river"*, and the answer D7, D9, D11, which is the intersection between the entry "logging" in Layer 1 and "river" in Layer 0. Finally, by specializing "logging" to "clearcuts", the query becomes *"clearcuts near river"*, and the answer is reduced to D11. Note that the same can be applied for generalizing concepts and enlarging the potential document set. Dynamically consulting the user on the appropriateness of proposed specializations or generalizations can also be done to enhance accuracy.

5 Assumption Grammars for Natural Language Processing

We shall now illustrate how Assumption Grammars can deal with inter-sentential dependencies through the example of anaphora, in which a given noun phrase in a discourse is referred to in another sentence, e.g. through a pronoun. We refer to the noun phrase and the pronoun in question as entities which co-specify, since they both refer to the same individual of the universe.

As a discourse is processed, the information gleaned from the grammar and the noun phrases as they appear can be temporarily added as hypotheses ranging over the current continuation. Consulting it then reduces to calling the predicate in which this information is stored.

We exemplify the hypothesizing part through the following noun phrase rules:

```
np(X,VP,VP):- proper_name(X),  +specifier(X).
np(X,VP,R):- det(X,NP,VP,R), noun(X-F,NP), +specifier(X-F).

pronoun(X-[masc,sing]):- #he.
pronoun(X-[fem,sing]):- #her.

anaphora(X):- pronoun(X).

noun(X-[fem,sing],woman(X)):- #woman.
```

The linear assumption, `+specifier(X)`, keeps in `X` the noun phrase's relevant information. In the case of a proper name, this is simply the constant representing it plus the agreement features gender and number; in the case of a quantified noun phrase, this is the variable introduced by the quantification, also accompanied by these agreement features.

Potential co-specifiers of an anaphora can then consume the most likely co-specifiers hypothesized (i.e., those agreeing in gender and number), through a third rule for noun phrase:

```
np(X,VP,VP):- anaphora(X), -specifier(X).
```

Semantic agreement can be similarly enforced through the well-known technique of matching syntactic representations of semantic types.

This methodology can of course be extended in order to incorporate subtler criteria. For instance, we can make each pronoun carry, at the end of the analysis, the whole list of its potential referents as a feature. User-defined criteria can then further refine the list of candidate co-specifiers, as in [6].

It is interesting to point out that in order to handle abstract co-specifiers [2], such as events or propositions, all we have to do is to extend the definition so that other parts of a sentence can be identified as possible specifiers as well. For instance, for recognizing "John kicked Sam on Monday" as the co-specifier of "it" in the discourse: "John kicked Sam on Monday. It hurt.", we can simply make the linear assumption that sentences are potential co-specifiers for pronouns of neuter gender.

Other important natural language processing problems (such as free word order) can be solved through hypothetical reasoning as well. The reader is referred to [7] for the details.

6 Assumption Grammars for Concept-Based Retrieval

Other than serving in the analysis of natural language sentences into meaning forms, as illustrated by the previous section, Assumption Grammars can contribute directly to concept-based retrieval, in conjunction with another system component we will describe in this section: the concept extractor.

Basically, the Concept Extractor takes an input such as keywords from the user interface or free text from the documents and extracts relevant words. These words are then used to determine the original text's relevance to concepts stored in the concept database.

The input can be parsed to remove unnecessary words. For example we may not need to deal with common words such as "the", "a", and "and". Once common words are removed, we are left with a set of distinguishing words that can be used to determine concept relevance and form a list of concepts-relevance pairs about the input.

The first step in being able to accomplish this, is to develop a concept "dictionary" or database. The entries in this database associate a concept with a list of words. It is these words that can be matched with the words extracted from the parsed input. For example, we could have the following entries in the dictionary:

- entry(("Natural Language Processing")-(parsing, generating, translating, speech)).
- entry(("Phonetics")-(speech, pronunciation, gaps))

Then we could take an input from either the user interface or a document and determine what concept the input is related to. The following could be an example input: *i need a document about translating speech to text.*

Now, let us assume that when this input is parsed, we have in our list of common words the following: (i, a, ",", ".", need, some, about). The result of the parse will be the relevant words from the input: (document, translating, speech, text)

We can then match these words to words in the database entries to determine concepts. In this case, we can determine that the example input is related to "Natural Language Processing" and "Phonetics". In the case that the relevant words from input match in two or more concept entries, we can return a list of concepts that the input is related to. But then we need to ask how relevant the input is to each of the possible concepts.

Relevance of input to concepts is a difficult task to deal with. As an example, imagine a document detailing the plumbing repairs in one's house. We run into the difficulty of determining if the document is more about plumbing, or house repairs.

As an attempt to deal with this difficult task, we associate the use of word count with amount of relevance. By doing this, we can use a function over word count and the words associated with individual concepts to evaluate the relevance of an input to the concepts. Further, we can introduce a logarithmic function so as to address the concern of documents or input containing numerous occurrences of particular words. This is a technique used by web designers to fool common search engines into listing the designer's web page as a highly relevant document by numerously adding a given word in a hidden META html tag.

Assumption Grammars can be used at this point to assist in creating a dynamic method for revising the concept hierarchies. As we can see in the above plumbing example, plumbing has a relation to house repairs. At this point, an assumption of the relation can be made. These relation assumptions can then be consumed later in order to revise the concept hierarchies.

7 Implementation

We have completed a preliminary implementation of a Concept Extractor specializing in computer science. The documents we used contain abstracts of technical reports published by the School of Computing Science at Simon Fraser University.

The translation of input words to concepts is accomplished through the same mechanism documents are categorized into concepts. As an example we can imagine a set of keyword inputs such as: *speech, parsing, translating* being translated into a concept list of: *Natural Language Processing*.

By being able to get a list of concepts from the user's input, we can advance the interface to include mechanisms for generalizing and specializing the concepts we are interested in. Through the use of concept hierarchies, as outlined in Section 4.4, we could possibly generalize "Database Systems" to "Computing Science" or we could specialize it to "Transaction Management".

This ability to use concepts to focus in (i.e. drill down) and out (i.e. roll up) of topics of interest to the user is advantageous. One advantage is that we indirectly guide the user to use concepts that are available through the system rather than having to worry about missing concepts or words in the lookup database. Secondly and maybe more importantly, we are able to use advanced natural language processing techniques to convert user input to a usable form for concept matching. An example here would be to use these advanced techniques to convert idioms such as "kicked the bucket" to the concept of "death".

8 Conclusion

We have described how a semantic-oriented analysis of both a www search query and a document that can potentially suit the query can benefit from our concept based approach and its related tools.

We are currently developing a prototype as proof-of-concept. This tool will prove useful for decision makers, as already stated, but also to all people with an interest in getting information from the web.

It is worthwhile mentioning that the use of the techniques here proposed will allow for relatively straightforward adaptations of our prototype from one domain into another, by replacing the concept hierarchy used for a given application domain by that required by another application domain. In this sense we expect the finalized system to be flexible enough for easy portability to different specific areas of expertise.

It is also interesting to note that semantic agreement constraints in the parser can also be used to disambiguate the text itself, thus avoiding the propagation of ambiguities over the search.

With the present work we hope to stimulate further research into what seems to be a very promising approach to concept extraction.

References

[1] H. Aït-Kaci, R. Boyer, P. Lincoln, and R. Nasr. Efficient implementation of lattice operations. *ACM Transactions on Programming Languages*, 11(1):115–146, 1989.

[2] N. Asher. *Reference to Abstract Objects in Discourse*, volume 50 of *Studies in Linguistics and Philosophy.* Kluwer, 1993.

[3] M. Bowman, P. Danzig, D. Hardy, U. Manber, and M. Schwartz. *Harvest, A scalable, Customizable Discovery and Access System.* Technical Report CU-CS-732-94 Department of CS, University of Colorado, Boulder, July 1994. Available from *ftp://ftp.cs.colorado.edu/pub/cs/techreports/schwartz/Harvest.FullTR.ps.Z.*

[4] V. Dahl. Incomplete types for logic databases. *Applied Mathematics Letters*, 4(3):25–28, 1991.

[5] V. Dahl, A. Fall, S. Rochefort, and P. Tarau. A hypothetical reasoning based framework for NL processing. In *Proc. ICTAI'96*, 1996.

[6] V. Dahl, A. Fall, and P. Tarau. Resolving co-specification in contexts. In *In Proc. IJCAI'95 Workshop on Context in Language*, Montreal, July 1995.

[7] V. Dahl, P. Tarau, and R. Li. Assumption grammars for processing natural language. In *Proceedings, ICLP'97*, 1997.

[8] B. A. Davey and H. A. Priestley. *Introduction to Lattices and Order.* Cambridge University Press, Cambridge, England, 1990.

[9] D. Eichmann. The RBSE spider - balancing effective search against web load. In *Proc. 1st Int. Conf. on the World-Wide Web*, pages 113–120, Geneva, Switzerland, May 1994.

[10] G. Ellis. Efficient retrieval from hierarchies of objects using lattice operations. In *Conceptual Graphs for Knowledge Representation. Proc. First International Conference on Conceptual Structures*, Quebec, Canada, 1993. Springer-Verlag.

[11] A. Fall. An abstract framework for taxonomic encoding. In *Proc. First International Symposium on Knowledge Retrieval, Use and Storage for Efficiency*, Santa Cruz, CA, 1995.

[12] A. Fall. The evolution of taxonomic encoding techniques. In *Estudios Sobre Programmación Lógica y Sus Aplicaciones*, pages 201–231. Servicio Publicacions da Universadade de Santiago de Compostela, 1996.

[13] A. Fall. Sparse term encoding for dynamic taxonomies. In *Fourth International Conference on Conceptual Structures*, Sydney, Australia, 1996. Springer-Verlag.

[14] J. Han, Y. Cai, and N. Cercone. Data-driven discovery of quantitative rules in relational databases. *IEEE Trans. Knowledge and Data Engineering*, 5:29–40, 1993.

[15] J. Han, Y. Fu, and R. Ng. Cooperative query answering using multiple-layered databases. In *Proc. 2nd Int. Conf. Cooperative Information Systems*, pages 47–58, Toronto, Canada, May 1994.

[16] J. Han, O. R. Zaïane, and Y. Fu. Resource and knowledge discovery in global information systems: A scalable multiple layered database approach. In *In Proc. Conference on Advances in Digital Libraries*, Washington, DC, May 1995.

[17] M. Koster. ALIWEB – archie-like indexing in the web. In *Proc. 1st Int. Conf. on the World-Wide Web*, pages 91–100, Geneva, Switzerland, May 1994.

[18] F. Mattern. Virtual time and global states of distributed systems. In *Parallel and Distributed Algorithms*, pages 215–226. Elsevier/North-Holland, 1989.

[19] M. L. Mauldin. Lycos: Hunting www information. CMU, 1994.

[20] O. McBryan. Genvl and wwww: Tools for taming the web. In *Proc. 1st Int. Conf. on the World-Wide Web*, pages 79–90, May 1994.

[21] C. Mellish. Implementing systemic classification by unification. *Computational Linguistics*, 14(1):40–51, 1988.

[22] G. Piatetsky-Shapiro and W. J. Frawley. *Knowledge Discovery in Databases*. AAAI/MIT Press, 1991.

[23] H. Jagadish R. Agrawal, A. Borgida. Efficient management of transitive relationships in large data bases, including is-a hierarchies. In *Proceedings of ACM SIGMOD*, 1989.

[24] R.L. Read, D.S. Fussell, and A. Silberschatz. A multi-resolution relational data model. In *Proc. 18th Int. Conf. Very Large Data Bases*, pages 139–150, Vancouver, Canada, Aug. 1992.

[25] S. Rochefort, V. Dahl, and P. Tarau. Controlling virtual words through extensible natural language. In *In Proc. AAAI Symposium on Natural Language Processing for the World-Wide Web*, Palo Alto, California, March 1997. (to appear).

[26] P. Tarau. BinProlog 5.25 User Guide. Technical Report 96-1, Département d'Informatique, Université de Moncton, April 1996. Available from *http://clement.info.umoncton.ca/BinProlog.*

[27] O. R. Zaïane, A. Fall, S. Rochefort, V. Dahl, and P. Tarau. On-line resource discovery using natural language. In *Proceedings, RIAO'97*, June 1997.

[28] O. R. Zaïane and J. Han. Resource and knowledge discovery in global information systems: A preliminary design and experiment. In *In Proc. 1st International Conference on Knowledge Discovery in Databases*, Montreal, Canada, August 1995.