EXTRACTING MEANINGFUL SEMANTIC INFORMATION WITH EMATISE: AN HPSG-BASED INTERNET SEARCH ENGINE PARSER

LIJUN HOU and NICK CERCONE

Computer Science
University of Waterloo
Waterloo, Ontario N2L 3G1 Canada

Abstract

We describe EMATISE, our prototype natural language interface that serves as a front-end to Internet search engines; this prototype is based on the HPSG (Head-Driven Phrase Structure Grammar) formalism. Our current work concentrates on parsing natural language phrases and extracting meaningful semantic information. We present the architecture of our system for internet access and the reasons for our choice of parser. The implementation of some modules of the system is discussed and some experimental results with sample sessions are presented.

Keywords

HPSG, natural language parsing, semantics, internet search engine, directories.

1 Introduction

The amount of information available on the world wide web is growing considerably, thus making retrieval of relevant information arduous. We provide natural language access to some of the major search engines and directories to enable people to access several catalogues for searching. The query is parsed and analyzed, resulting in the selection of relevant keywords for search. Using relevant keywords significantly reduces the listing of irrelevant sites. Synonyms are searched automatically as applicable.

Search engines and Internet directories are primary vehicles that Internet information seekers use to find web sites and documents of interest. Most current search engines (spider, index and search engine software) use search strategies based on keyword matching. Some search engines allow conjunctions of keywords, combinations of disjunctions of keywords, and negations. The Boolean combination of keywords is evaluated and matched with the words in an inverted index, which is built in advance, to retrieve documents containing these keywords.

Users may become frustrated by not knowing how to pick appropriate keywords to formulate query statements. Users may not find what they want in such on-line searches because the words they use fail to match words that are needed in the documents. Also, casual users may have problems with the subtleties of Boolean logic, which is used by most search engines to combine keywords into a query statement. Table 1 illustrates how crucial is the choice of keywords to determine search engine response. Yet there appears little consistency in keyword interpretation within a search engine or between search engines.

Several search engines purport to use natural language for access. It appears that their attempt to provide better ad-hoc access was not totally successful as Table 2 illustrates. In Table 2, natural language phrases were entered in an attempt to find the "best" or "top" computer science "department" or "program". The results indicate that semantic analysis was either not performed or performed inadequately.

Each Internet "engine" has a listing or catalogue to search. Depending on how URLs (uniform resource locators) are added to catalogues, they are classified as a directory [Yahoo Home Page. Nov. 2000. http://www.yahoo.com] or as a spider or robot [AltaVista home page, Nov. 2000, http://www.altavista.com/; Info-seek home page, June 1997, http://www.infoseek.com; Lycos home page, Nov. 2000, http://www.lycos.com]. We refer to both types as search engines unless noted.

Typically, keyword searches exhibit the following problems:

- Since a keyword can have many synonyms, all synonyms need to be used in the search in order to obtain all relevant sites.
- Variations in spelling might result in missing some of the relevant sites. A search for color might miss a site with the word colour and a search for gray will not find grey, and so on.
- Ambiguity in selection and ordering of keywords might result in different search results when a mere word match is performed. For example, if a search is made for sites performing online reservation of flight tickets, the keywords can be any of the following: (1) reservation, flight, ticket; (2) online reservation, flight, ticket; (3) online reservation, air, ticket; (4) online reservation, flight OR air, ticket; or (5) reservation OR booking, flight OR air, ticket.

0-7803-7087-2/01/$10.00 © 2001 IEEE

Each of the sets of keywords in combination with options exclusively supported by each search engine like "+" and "-" might be tried by the user, as each of them yields different results both in terms of number of sites returned and in their ordering in the results returned.

Keywords	Infoseek	Yahoo	Lycos
top, computer science	4,606,219	7	17,374
+top, computer science	78,414	3071	17,374
top, +computer science	76,989	1064	17,374
+top, +computer science	2,132	7	17,374
top, computer science department	4,606,545	2	28,403
+top, computer science department	78,386	3071	28,403
top, +computer science department	76,991	101	28,403
+top, +computer science department	2,131	2	28,403
top, computer science program	4,606,672	1 (RHIT)	23,925
+top, computer science program	78,415	3071	23,925
top, +computer science program	76,991	1(N.Ga.)	23,925
+top, +computer science program	2,132	162	23,925
rank & computer science program	7,570,132	16460	29,205
rank & computer science department	4,604,421	133,690	28,694
rank and computer science department	23,267,722	133,690	28,694
rank I computer science department	4,117	133,690	28,694
rank or computer science department	6,070,374	133,690	28,694

Table 1. Choice of keywords is crucial to search engine response.
Note: "+" is used to indicate that the word following must appear for the search to be successful; "&" and "|" are logical operators

Search engines that employ a concept search look for pages containing the exact query words entered or also for concepts closely linked to the query words. This feature of concept searching broadens the search, e.g., a search for elderly people might bring up a site that only has retired people or senior citizens [Excite Home Page. November 2000. http://www.excite.com]. However, the selection of keywords plays an important role in the search results obtained, as in the case of keyword searching. For example selecting appropriate keyword(s) may be a difficult task for searches like airlines flying between Canada and Japan.

Selecting appropriate keywords for a distributed information resource like the WWW is difficult, even for an expert. Hence many search engines are now providing phrasal and natural language search options. Some queries are better expressed as phrases rather than keywords. The possibility of finding the most relevant site as a top ranking one is higher in these search engines when compared to keyword matching. Most existing natural language search engines are restrictive and allow users to present queries as phrases or questions alone. The following searches are easier to express as sentences than as questions: "I would like to travel Europe by rail" and "I want to stay in Vancouver for two days". Moreover, the popularity of these search engines depends on other features like response time, catalogue updates, and relevancy ranking.

natural language	Number of hits Infoseek, 7/1997	Number of hits Lycos, 11/2000
Which is the best computer science department?	22,497,252	276,497
Which is the best computer science program?	22,497,257	344,642
Which is the top computer science department?	22,497,276	240,964
Which is the top computer science program?	22,497,290	295,051
the best computer science department?	22,497,339	326,032
the best computer science program?	22,497,350	428,316
the top computer science department?	22,497,299	308,597
the top computer science program?	22,497,305	399,084
best computer science department?	4,931,348	331,822
best computer science program?	7,895,146	428,927
top computer science department?	4,681,660	322,200
top computer science program?	7,653,131	407,306
computer science department?	4,667,918	1,086,127
computer science program?	7,638,548	1,224,053
computer science?	4,665,750	3,859,408
computing science?	2,588,408	857,813

Table 2. Attempt using natural language access.

Although there are many search engines, none has been objectively judged to be better than the others are. Some search engines are more suitable for searches made on general topics or categories, some may have more features for searching than the others, some may list matches differently, some may return search results more quickly and so on. Hence, the correct choice of the search engine will depend on the specific search requirements of the user.

We developed a natural language interface [NLI] to serve as the front-end to Internet search engines thus enabling the user to access several catalogues for searching. The NLI uses natural language processing techniques to eliminate the need for formal query syntax and to permit free-language query submissions so as to assist casual users find the information they seek.

NLI's have been built for databases in the past, see (Cercone and McCalla 1986) and (Perrault and

Grosz 1986) for a detailed overview of NL database interfaces, their problems and solutions which are embodied in the many actual systems they discuss. Our NLI is different from conventional NL database interfaces since the WWW is a collection of heterogeneous information and of many different modalities. Query languages are different and search strategies vary.

Our prototype is implemented using Sicstus Prolog and Attribute Logic Engine [ALE] (Matheson 1997), which is an integrated phrase structure parsing and definite clause logic programming system based on the HPSG formalism (Sag and Wasow 1997). The system also uses OpenText as the target search engine, and its manufacturing slice of computer products as one of several experimental search domains. System modules and their configuration are shown in Figure 1.

The user interface module accepts natural language queries from the user and translates this input into a Prolog list of atoms that is required by the parser. The parser module generates all valid parses for the input list. The semantic extractor converts the attribute value matrix [AVM] feature structure, the result of parsing, into the logical query (term expansion) that is acceptable to the search engine. The search engine access module passes the logical query to the OpenText search engine and provides the user with results from the natural language query.

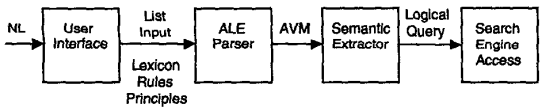


Figure 1. System Configuration.

We briefly explain the operation of the parser and the semantic extractor, together with a working lexicon in the next section, followed by our experimental results.

2 HPSG Parser

HPSG is a constraint-based, lexicalist approach to grammatical theory that models language as a system of constraints built upon typed feature structures (Pollard and Sag 1994; Sag and Wasow 1997). Linguistic information is represented in a multiple inheritance hierarchy of feature structures. A feature structure, or AVM (attribute-value matrix), consists of a set of attribute/value pairs, which store syntactic and semantic information about words, and phrases, see in Figure 2.

sign		
SYNSEM	synsem	
LOC	loc	
	CAT	cat
	HEAD	head
	MARKING	marking
	SPR	synsem_list
	SUBCAT	synsem_list
CONT	cont	
	SEM_MARK	dconcept
	QUERY	query_list

Figure 2. Feature Structure.

A sign is a lexical entry, a phrase or a sentence. It has the feature SYNSEM for storing both syntactic and semantic information. SYNSEM, in turn, has a feature of its own, called LOC, containing local information, while nonlocal information needed to compute long distance dependencies is not implemented in our prototype. The LOC feature introduces two features CAT, which contains the syntactic information (e.g., HEAD, MARKING, SPR and SUBCAT) for the head of the sign, and CONT, which contains semantic information for the sign.

CONT is comprised of features SEM_MARK and QUERY. SEM_MARK is used for selectional restrictions and to control ambiguity. QUERY contains a list of query terms. Each term is composed of the following four features: ACTION, KEYWORD, SYNONYMS and ANTONYMS to capture the semantic interpretation of the query terms and the logical relationship among them.

HPSG is a highly lexical grammar formalism and its lexicon is rich in information. Only a small set of rules and principles are introduced to control the joining of syntactic constituents, and they handle a wide variety of English constructions.

The grammar rules used in HPSG control how the different classes of syntactic constituents are joined to form larger grammatical structures. In our prototype, four grammar rules are defined in HPSG: Subject-Head, Head-Complement, Head-Subject-Complement and Adjunct-Head. They are used together with two additional rules: Specifier-Head and Conjunct rules, to handle specifiers and conjunctions.

Each of the rules stated above is processed in conjunction with a collection of principles. Our prototype uses five universal principles outlined in HPSG: Head feature principle, Subcategorization principle, Specifier principle, Marking principle and the Semantic principle.

The semantic principle unifies the CONT feature of signs. In our prototype, it is especially tuned to be able to handle the unification of semantic information for query terms in search engine queries. The semantics principle combines the SEM_MARK and QUERY information for the words or phrases being joined.

The action principle is defined to handle the inter-relationship between the semantic information of

the query terms. This principle ensures that the logical conjunction of the daughter is assigned to be one of and, or, not.

For additional details regarding this prototype HPSG parser, see Hou (1999).

3 Semantic Interpretation and Semantic Extractor

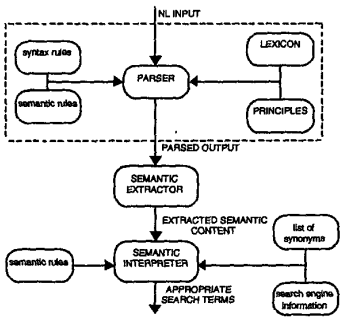
Traditional parsers use a set of syntactic grammar rules along with a lexicon to parse the natural language. Applied to the parsed results, semantic rules convert the parse tree into a logical query. We carry out semantic analysis in two phases. Since an HPSG parser is used, the parser does critical semantic analysis during parsing. Further detailed analysis is performed in the semantic interpreter as illustrated in Figure 3.

All successful distinct parses generated by the parser are transmitted to the semantic extractor for further processing. The semantic extractor is a Prolog program that converts the output of the HPSG parser from a complex AVM feature structure into a logical query. The logical query is term expanded for the query language accepted by the target (OpenText) search engine

The list of keywords from the extractor are converted into appropriate search terms using the semantic rules, synonyms list and search engine information. Semantic rules are applied primarily to group nominal compounds. For example, in South Africa, the word south would be treated as an adjective in the lexicon. However South Africa has to be present as

Figure 3. Two Phases of Semantic Analysis.

the name of the country in the search term. In order to ensure that no relevant sites are missed, all of the synonyms of keywords, wherever applicable, are included in the search terms using the boolean option 'OR'. Search engine information provides details of search options supported by individual search engines so as to map the search terms with the search engine selected. While doing so, the interpreter identifies keywords that must be present in the sites listed



and encodes them with signs like '+'.

OpenText's text search product Livelink Pinstripe provides slice search to ensure highly targeted searches for business users. There are over 150 precategorized information groupings, each grouping representing a search slice on a particular business topic. Its computer and electronic product manufacturing slice contains information on computer manufacturing, communications equipment, audio and video equipment, semiconductor manufacturing, control instrument manufacturing, etc. Our prototype used their computer manufacturing search domain.

Each resulting AVM returned by the parser is represented by ALE internally as a nested Prolog list structure. This list takes the form of phrase(-synsem(-loc(-cat,-cont))), where cat encodes the syntactic information, and cont conveys the semantic information. As an example, the list structure resulted from parsing the phrase "computer makers" is shown in Figure 4. The semantic extractor first extracts the CONT value from the AVM. Then the QUERY value is, in turn, extracted from the CONT value. Other information which is not presently used in the semantic conversion, e.g., the CAT and SEM_MARK values, are just discarded.

```
phrase
SYNSYM  synsym
        LOC loc
            CAT cat
                HEAD  noun
                      AGR   agr
                            NUM      plural
                            PERSON   PERS
                      CASE  case
                      MOD   none
                MARKING marking
                SPR     synsem_list
                SUBCAT  e_list

CONT    cont
        QUERY ne_query_list
              HD  query
                  ACTION      conj
                  ANTONYMS    antoynm_list
                  KEYWORD     computer
                  SYNONYM     synonym_list
              TL  ne_query_list
                  HD  query
                      ACTION      conj
                      ANTONYMS    antoynm_list
                      KEYWORD     company
                      SYNONYM     ne_synonym_list
                                  HD_manufacturer
                                  TL   e_list
                  TL  e_list
        SEM_MARK  dconcept
```

Figure 4. Example of the AVM Feature Structure.

Each element in the list value of the QUERY feature is analyzed to obtain the query terms. While extracting information from the QUERY list, a structure is created to store information about the current query. This structure is a list of predicates, with each predicate containing query information for a single query term. The symbol of the predicate in the list is the logical conjunct (e.g., and, or, not) extracted from the element of the QUERY list. If the logical conjunct is unspecified, then it takes the default value of

"and". Each predicate takes three arguments. The first argument is the KEYWORD value extracted. The second argument is a list containing synonyms of the KEYWORD, each synonym is extracted separately and concatenated to the list. The third argument is a list containing antonyms extracted for the KEYWORD. An example structure for the phrase "computer makers" follows:

[and (computer, [ ], [ ]), and (company, [manufacturer], [ ])]

After the structure has been completely built from the AVM, it is passed to the predicate that handles printing of the logical query. The elements in the list value of the structure are processed and printed individually. If the KEYWORD value of an element is unspecified, then it represents a general concept that does not contribute to the selection of a query term, thus it is ignored. The output of a query term depends on the logical conjunct associated with it.

For the logical relationship of negation (e.g., not), the query term is composed of the list of antonyms of the KEYWORD, if any. An example logical query output for the phrase "less expensive computers" is:

and (economic sor cheap) and computer

If the KEYWORD has no antonyms, the domain concept hierarchy will be examined to look for any sibling concept of the concept represented by the KEYWORD. Such an example would be for the phrase "computer companies outside Canada", its logical query output is:

and computer and (company sor manufacturer) and (usa sor europe)

If the logical relationship is anything other than negation (e.g., and, or), then the query term is composed of the KEYWORD and the list of its synonyms, if any. An example logical query for the phrase "computer makers" is:

and computer and (company sor manufacturer)

If the KEYWORD has no synonyms, the domain concept hierarchy will be examined to look for any child concept of the concept represented by the KEYWORD. An example logical query for the phrase "computer parts" is:
and computer and (hardware sor desktop sor notebook sor server sor monitor
sor modem sor printer sor scanner sor camera sor memory sor software)
If there exists no child concept, the query term is the KEYWORD itself.

4 Generic Lexical Processing

The lexicon plays an important role in HPSG grammar. Most importantly, it defines a finite set of lexical signs (words) permitted as input. The ALE parser works bottom--up from information in lexical entries to unify their feature structures. The parse will fail immediately as soon as it finds any word in the user query that is not defined in the lexicon. However, as part of the natural language processing module of an Internet meta search engine, the lexicon will almost always suffer from this incompleteness, even for restricted domains. This situation is especially true for the case of proper names. Proper names could be found making up a large portion in Web queries; with a name being either a person or an organization's name, or any name related to an event or an object, etc. Despite their pervasive use in queries, proper names seem elusive and unpredictable. It will be almost impossible trying to include all or most of them into a finite lexicon. We discuss several different methods, which could be used to solve this problem, with the emphasis on the approach used in our parser as a partial solution to this problem.

When faced with this challenge, two kinds of solutions are arrived at easily. The first solution is to simply reject an input sentence, which contains any undefined words, and let the parsing fail. The second solution is to probe users with questions, to acquire lexical knowledge about the undefined words "on the fly", dynamically expanding the lexicon as the parser learns more words.

Neither of these two methods above are desirable, since neither method is user--friendly, either by rejecting the user request without making use of any other information that might be available, or burdening the user to specify a full definition of the undefined proper name which might scarcely be used in the future.

We propose an approach which defines a special generic lexical entry in the lexicon that takes on the properties of a proper name and substitutes any undefined words in the input by this generic lexical entry before parsing. If its original syntactic role is indeed a proper name, then parsing succeeds. From the parsed output, we replace the generic lexical item by the original words in the input, and then send it for Web searching. But if it does not fit syntactically into the role of a proper name, the parsing will fail in the unification process.

The basic idea for introducing such a generic lexical entry is to make it serve as a placeholder for those words undefined during parsing. Since the undefined proper nouns do not actually participate in the parsing process, they are not analyzed semantically by the system; for example, the parser will not work out any synonyms or antonyms in terms of its word relationships. But this kind of processing fits well with the linguistic nature of proper nouns, since it is usually not necessary for this type of semantic analysis to happen for proper nouns.

By including a generic lexical entry in the lexicon and having it act as the intermediary for undefined proper nouns, we have partially solved the problem of lexicon incompleteness for the case of proper names in Web searching. This effectively improved usability of the NL processing system by giving it power to process a much larger amount of words than the fixed lexicon is predefined to handle.

5 Other Parsing Considerations

There are many other linguistic issues that exist in the development of a natural language processing system. For example, when interpreting a user's query and converting it into logical form, the intrinsic subtleties of Boolean logic with negation, conjunction, and disjunction might add to the semantic ambiguity of a query. In the phrase "papers written by Nelson and Woods", the "and" normally does mean the logical "AND". But when stating "travel agents in Canada and the US", the Boolean expression with "and" actually means the logical "OR". There are also other cases where the "or" could be misunderstood to mean the "exclusive OR", etc.

Another example of a linguistic issue with an important impact on natural language understanding is how to translate a user's query when the query contains specifications such as "frequent, very, recent", etc. In the phrase "recent models of Honda", "recent" may take on "years" as the chronological measure. Furthermore, "recent" may mean different things in 1996 than in 1999. In the phrase "recent data on currency exchange", "recent" may mean "months" or even "days".

The goal of Ematise is to present a working prototype which uses recent advances in HPSG grammar formalisms and to experiment with some crucial aspects of natural language processing for which these advances are particularly well-suited. Thus, not all linguistic issues are covered, and not all solutions are incorporated into Ematise.

6 Multi-Domain Semantic Analysis

In order for the natural language interface to be of practical use to query the vast collection of data that resides on the Internet, the parser must be capable of extending its coverage over multiple search domains. However, when multiple domains are present, there could exist potentially different semantic interpretations in different contexts for a single query. We discuss the impact of the increased complexity of the parser due to semantic ambiguity and outline the approach our parser uses to resolve ambiguity in a multi--domain query environment.

The Ematise parser defines the feature named SEM_MARK (Semantics Marker) to contain the context information for a phrase or sentence. In the implementations of general principles, especially the Semantics Principle, which constrains the unification process for semantic information, the semantic interpretation of the query terms only are successfully unified into the query list if their SEM_MARK information is unifiable. This is the basic idea used to disambiguate sentences or phrases among different contexts and domains. When the parser cannot resolve ambiguity, the system generates all valid parses for the input, and prompts the user to choose among alternatives to resolve the ambiguity. For example, a user may enter on--line query requests such as:

[1] I want to visit Japan.

[2] I want to visit the homepage of IBM product reviews.

In sentence [1], "visit" has a particular semantic meaning for "travel", while in sentence [2], it does not convey any specific semantic interpretation in our system. This ambiguity can be handled by defining multiple entries in the lexicon for the word "visit", represented by feature structures as shown in Figure 5.

```
(a)  word      visit
     SYNSEM | LOC | CAT |  HEAD      verb
                          SUBCAT    <NP[SEM_MARK 1 ]>
                  CONT    SEM_MARK  location 1
                          QUERY     <travel>

(b)  word      visit
     SYNSEM | LOC | CAT |  HEAD      verb
                          SUBCAT    <NP [SEM_MARK 1 ]>
                  CONT    SEM_MARK  dconcept 1
                          QUERY     < >
   "dconcept" is the default value for feature SEM_MARK
```

Figure 5. Feature Structure for "visit".

In our example sentence (1), the feature structures for the noun "japan" conveys "location" as its value for the SEM_MARK feature, see Figure 6.

```
word      japan
SYNSEM | LOC | CAT |  HEAD   noun

                CONT    SEM_MARK  location
                        QUERY     <japan>
```

Figure 6. Feature Structure for "japan".

Since "japan" is unifiable with the top feature structure defined for "visit", the phrase "visit japan" can be unified based on the Head--Complement schema, with the resulting feature structure after unification as shown in Figure 7.

```
Phrase    visit    japan
SYNSEM | LOC | CAT | HEAD        verb

                CONT  SEM_MARK  location
                      QUERY     <travel, japan>
```

Figure 7. Feature Structure for "visit japan".

For our example sentence [2], the noun phrase "the homepage of IBM product review" does not have "location" as its SEM_MARK, so it can only be unified with the second feature structure defined for the word "visit", in which case "visit" does not contribute to a specific semantic interpretation.

7 EMATISE

The meta search engine of Ematise is designed in modular fashion, as depicted in Figure 8. The main components are the CGI interface, the aggregation engine, and the search service drivers. The CGI interface is simply a layer that passes user's query option in a logical format which is search service neural from Web client to the meta search engine server. The logical query is then passed on to the aggregation engine, which is responsible for concurrently dispatching the query to selected search services, obtaining the initial results from each service, eliminating duplicate results, consolidating and re-ranking the results, and finally creating HTML pages from the results, to be properly displayed back at the Web client. The individual search service drivers are responsible for translating the original logical query which is in a search service neural format into some service specific query, sending the query to the service, and receiving results. These drivers are implemented as a collection of independent Java classes, where each class represents a particular service. It is designed so that classes can be added, modified and removed without impacting the rest of the meta search engine.

CGI Interface and Client-Server Communication
The meta search engine is a Java application on the server. The Web client which is developed in Java applet communicates with the meta search engine on the server side through a small CGI interface written in Perl. For each user query, the applet makes a CGI call to the server, passes the user query request to the CGI, which in turn invokes the Java application on that same server for the meta search. The meta search engine resides separately on the server, and is not fully integrated with the Web client because of the strict security restrictions enforced on Java applets. The meta search engine must establish URL connections with selected search engines over the Internet, this can only be implemented in a standalone Java application on the server, since applets are not permitted to have network connections with servers other than the one from which the applet is invoked. This restriction is one of the safeguards on untrusted applets, which blocks some of Java's functionality. The loss is a trade-off for the security that must be in place for the language to run remotely on Web client's computers.

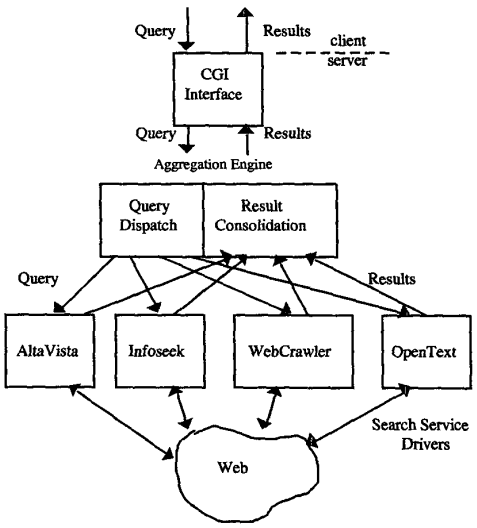


Figure 8. Meta Search Engine Architecture.

Of course, in addition to CGI, this client-server invocation could also be achieved through other alternative options, such as using Java RMI (remote method invocation) package.

The aggregation engine is one of the major components of the meta search engine for EMATISE. The algorithm and control for the aggregation engine is discussed in detail in Hou (1999).

Search Service Drivers are a collection of Java classes; each executes in a separate thread for a particular search service. They contain all the service specific information. In Ematise, we selected four popular Internet search services: AltaVista, Infoseek, WebCrawler, and OpenText. Based on their index size, which is regarded as a clear indication of how comprehensive a record of the Web the search service covers, the first three are categorized into big, medium, and small search services, respectively. OpenText is a specialty search service characterized as a business oriented search service. Table 3 summarizes some of their features.

Search Service	Index Size (millions of web pages)	http Method	Term In-clude/E-xcude	Phrase	Boolean	Limit on # of terms
AltaVista	150	GET	Yes	Yes	Yes	No
Infoseek	45	GET	Yes	Yes	Yes	No
Web Crawler	2	GET	Yes	Yes	Yes	No
OpenText	5	POST	No	No	Yes	Yes

Table 3. Comparison-Meta Search Engine Properties.

After translating and sending the query with the proper format to a designated search service, the driver must also receive and parse the query results from that service. Each service could return its results in a format that is completely different from others. The driver class has to understand the output format and correctly parse the output to extract the information of the returned documents. While gathering returned results, any duplicate documents returned by the same service are simply removed.

The meta search engine of Ematise provides a layer of abstraction above traditional search services. Its design incorporates several desirable features.

As the Web grows and changes, search services are also volatile in nature. New search services are being launched continually. The search interface of existing search services also changes quite often due to enhancements or upgrades which might have an impact on both its query input and output format. Also there are a number of services being retired or replaced. The modular design of the meta search engine of Ematise, especially the collection of search service driver classes, provides a wrapper around this service specific information, effectively encapsulates them, and allows for services to be added, modified, and removed easily and cleanly, thus providing good adaptability toward a changing Internet.

Ematise has a meta search engine that does not require large databases or large amounts of memory by itself. Since both the server and client side of the meta search engine are implemented in Java, they are easily portable to different platforms without the extra effort of changing the code.

8 Experimental Results

To experiment with the target search domain, computer manufacturing, a small HPSG lexicon with over 200 words was built, together with the typed feature structures, grammar rules and universal principles outlined above. The lexical entries are organized in inheritance hierarchies based on their semantic groupings. Words are classified into top level semantic categories, which include: computer components, companies, places, proper nouns, nouns, etc.

Suppose a user wants information on "computer makers outside of Canada". Parsing of this natural language phrase contains the following QUERY feature as the output of unification, see Figure 9.

Each keyword from the query list is extracted and analyzed. To reflect the interrelationship of the keywords, a logical structure is constructed:

[and (computer,[ ],[ ]), and (company,[ ], [manufacturer]), not (canada,[ ],[ ])]

Here the word "maker" contributes to keyword "company" or its synonym "manufacturer". The word "outside" is interpreted as its logical equivalence "not", and is unified with the ACTION feature of the head of the noun phrase following it, to form the predicate "not (canada,[],[])". By examining the concept hierarchies of the keywords, this predicate translates into "(usa sor europe)", the synonym-or of the two sibling concepts of the key concept {\it canada} in the hierarchy. So the final output from the natural language processing module is the following logic query that is compatible with OpenText search syntax:

and computer and (company sor manufacturer) and (usa sor europe)

```
QUERY   ne_query_list
        HD      query
                ACTION conj
                ANTONYMS   antonym_list
                KEYWORD    computer
                SYNONYMS   synonym_list
        TL      ne_query_list
                HD   query
                     ACTION [0]
                     ANTONYMS   antonym_list
                     KEYWORD    canada
                     SYNONYMS   synonym_list
                TL   ne_query_list
```

Figure 9. The Output of QUERY Feature.

We have integrated our natural language interface with OpenText's search engine to provide a realistic environment for the experimentation and evaluation of retrieval effectiveness. For example, for the above listed user query, the test against OpenText without the natural language interface actually yields no documents. However, if we query against OpenText via the natural language interface, the semantic meaning of the input is analyzed and properly extracted, the same query yields 89,431 Web documents. By picking samples at random among the returned Web documents, we found around 49% of them were relevant to the query. Our general experiments show that the natural language interface effectively improves the quality of search engine results (Mahalingam and Cercone 1999).

A few sample screenshots illustrating several examples are given in Figures 10-12. Figure 10 shows a simple translation of the sentence "I want to visit the homepage of ibm product review" into search engine neutral search terms to be term expanded by the drivers for particular search engines. Figure 11 illustrates the results of this query after the results are assembled by the aggregation engine. Figure 12 illustrates another use of the verb "visit", as "stay", for the query "I want to stay cheaply in Vancouver".

9. Concluding Remarks

An Internet search engine can provide enhanced performance by integrating conventional search engine technologies with techniques adapted from natural language processing. A search engine with a NL interface could be more user-friendly and precise, by actually eliminating the need for a formal query-

formatting syntax and the formulation of a request as logical manipulations of specific keywords.

A first prototype of such a natural language interface system was built with current concentration in natural language parsing and semantic extraction. Our purpose is to make a comparison of the effectiveness of retrieval by syntactic and semantic analysis of phrases and retrieval by simple keyword-based approaches, and to evaluate any potential improvements introduced by natural language analysis.

An important aspect of our future work is to explore the use of multiple search domains, which not only requires expanding the lexicon, but also involves research on context analysis.

Acknowledgments

The authors are members of the Institute for Robotics and Intelligent Systems (IRIS) and wish to acknowledge the support of the Networks of Centers of Excellence Program of the Government of Canada, the Natural Sciences and Engineering Research Council, and the participation of PRECARN Associates Inc.

References

Cercone, N, and McCalla, G. (1986) Accessing Knowledge Through Natural Language. Invited chapter for M.Yovits 25th Anniversary Issue Advances in Computers series, Academic Press, 1-99.

Hou, L. (1999). EMATISE: English Meta Access to Internet Search Engines. M.Math Thesis, Computer Science, University of Waterloo, Waterloo, ON., Canada.

Mahalingam, G., and Cercone, N. (1999) Finding Information Easily is Important for e-Business!, Data Warehousing and Data Mining for Electronic Commerce W. Kou (ed.), IBM Press, 135-168.

Matheson, C. (1997) HPSG Grammars in ALE. Centre for Cognitive Science, University of Edinburgh. http://www.ltg.hcrc.ed.ac.uk/projects/ledtools/ale-hpsg/.

Perrault, C. R., and Grosz, B. J. (1986) Natural Language Interfaces. Annual Review of Computer Science. Palo Alto, California, USA.

Pollard, C., and Sag, I. (1994) Head Driven Pharse Structure Grammar. Univ. of Chicago Press.

Sag, I., and Wasow, T. (1997) Syntactic Theory: A Formal Introduction. http://hpsg.stanford.edu/hpsg/sw-tb.html.

Figure 10. Input to Ematise of "I want to visit the homepage of ibm product review".

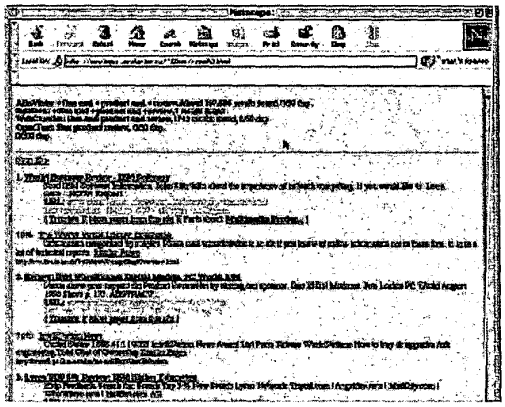


Figure 11. Output from Ematise for "I want to visit the homepage of ibm product review".

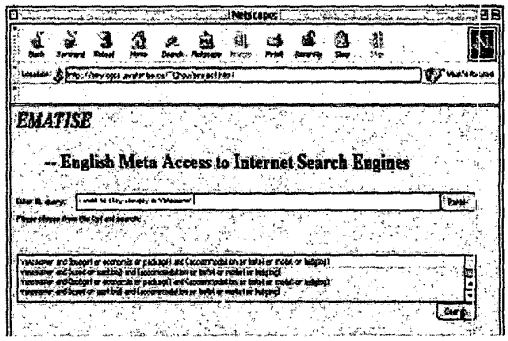


Figure 12. Input to Ematise for "I want to stay cheaply in Vancouver".