Språkbehandling och datalingvistik

Projektarbeten 2001

Handledare: Pierre Nugues

LUNDS UNIVERSITET

Institutionen för Datavetenskap

http://www.cs.lth.se

Printed in Sweden
Lund 2004

Innehåll

Mathias Haage, Susanne Schötz, Pierre Nugues:
A Prototype Robot Speech Interface with Multimodal Feedbac
Reprint from Proceedings of the 2002 IEEE Int. Workshop on Robot and Human Interactive Communication Berlin, Germany, Sept. 25-27, 2002

LUNDS UNIVERSITET

Institutionen för Datavetenskap

http://www.cs.lth.se

Identifying Collisions in NTSB Accident Summary Reports

Torbjörn Ekman
Anders Nilsson
Dept. of Computer Science, Lund University
Box 118
S-221 00 SWEDEN
Tel: +46 46 222 01 85
Fax: +46 46 13 10 21
email: {torbjorn|andersn}@cs.lth.se

June 13, 2002

Abstract

Insurance companies are often faced with the task of analyzing car accident reports in order to try to find out how the accident took place, and who to blame for it. The analysis could in many cases become much easier if the accident could be automatically visualized by extracting information from the accident report.

This paper describes the collision detection functionality of the text processing system that is used with CarSim, a car accident visualization system. Using Link Grammar and regular expression pattern matching, we have correctly extracted 60.5% of the collisions found in 30 authentic accident reports.

1 Introduction

CarSim [DELN01] is system for visualizing car accidents from natural language descriptions developed at the University of Caen, France. The aim of the CarSim system is to help users to better understand how a car accident took place by visualizing it from the description given by the participants in natural language. The visualization process is performed in two steps. First comes an information extraction step where the natural language texts are analyzed, relevant information is extracted[1] and templates are filled with the extracted information. The second step reads templates and constructs the actual visualization.

As part of further improvements in CarSim, a new information extraction system for English texts is being developed at the Department of Computer Science, Lund University. This paper describes how collisions are detected, and how to find extract collision verbs together with their subject/object pairs. In some cases it is also possible to resolve co-references in the analyzed text.

[1] Not only the collision as such, but also other vehicles, trees, rocks, road signs etc.

1.1 NTSB Accident Reports

The National Transportation Safety Board (NTSB) [NTS] is an independent federal agency in the United States investigating every civil aviation accident in the US, and also significant accidents in other modes of transportation, such as highways and railroads. The aim is to understand why the accidents occurred and issue safety recommendations to prevent similar future accidents.

The NTSB publishes accident reports, and for many of them, shorter summaries, describing investigated accidents. The accident reports summaries, were used to test the implemented collision detector.

1.2 LinkGrammar

In the beginning of the 90's, Sleator and Temperly [ST] defined a new formal grammatical system they called a *link grammar*, related to the larger class of dependency grammars. The idea behind *link grammar* was to let the words of a sentence create a connected graph with no crossing arcs. The arcs connecting words are called links. These links describe the syntactic and semantic relationship between the connected words, i.e. connect a determiner to a noun or a noun phrase to a verb.

1.3 WordNet

WordNet [Fel] is an online lexical reference system whose design is inspired by current psycholinguistic theories of human lexical memory. English nouns, verbs, adjectives and adverbs are organized into synonym sets, each representing one underlying lexical concept. Different relations link the synonym sets. WordNet was developed by the Cognitive Science Laboratory at Princeton University.

WordNet is in this project used for extracting all possible collision verbs, see section A, that could occur in the accident reports.

WordNet is currently being adapted to other languages, notably Swedish at the Linguistics Department at Lund University.

2 Subject and Object Detection in Collision Context

The subject and object detection is divided into three stages. First, a shallow analysis locates sentences that may describe collisions. That analysis uses regular expressions to find the texts containing collision verbs. These sentences are passed on to Link Grammar which performs a much more in depth analysis. The subject and object are finally detected in the link representation of a sentence by using a set of patterns describing how to locate those entities starting out with a collision verb.

2.1 Detecting Sentence Candidates

Using Link Grammar to analyze sentences is a quite costly operation. Although the accident reports are fairly short it is not very practical to analyze all sen-

tences when detecting collisions. Therefore an initial pass to collect a set of candidate sentences suitable for deeper analysis is performed.

The accident reports are first tokenized to form a set of sentences. Each sentence is then matched against a regular expression accepting sentences including collision verbs. Each accepted sentence is passed on to Link Grammar, a much deeper analysis engine.

The regular expression used is built to accept a sentence including any collision verb from a list of verbs. A complete list of collision verbs, extracted from WordNet, is available in Appendix A.

2.2 Linkage Patterns

The result from the Link Grammar analysis is a graph where a word has a number of arcs connected to it. Each arc has a direction, left or right, as well as a semantic meaning. The transition from one word to any connected word can be expressed as a series of link and word pairs, where the link has a direction and a type, and the word has a name and a type. We call such an ordered list of pairs a Linkage Pattern.

Our analysis engine processes a Linkage Pattern where each pair is expressed by a regular expression. This way a large set of link-word pairs may be expressed through a compact notation. The engine matches the regular expressions and returns the set of words that is reached by following the entire pattern. The regular expressions is used by the system to locate subject, object, and also to resolve a class of co-references.

2.3 Extracting Subject and Object

Starting out from the collision verb a set of Linkage Patters are matched against the Link Grammar representation of the candidate sentences. These Linkage Patters describe how to reach the subject and object when starting on the collision verb. In its simplest form, a pattern may find the subject by following a Subject link to the left from the collision word.

2.4 Handling Co-references

The Linkage Pattern technique can also be used to resolve co-references. In this case, a pattern describes how to reach the subject starting with a pronoun. Because Link Grammar only processes a single sentence, the subject and pronoun must be in the same sentence for this technique to be successful.

3 Result Analysis

The collision detector was run on 30 NTSB accident report summaries containing a total of 43 collisions. The result is shown in table 1, and the individual results for each collision report with a short comment can be found in appendix B.

Analyzing the erroneous and incomplete collision detections confirms the suspicion that the regular expression pattern matching works correctly, but that

Correct detections	26
Erroneous detections	5
Incomplete detections	12
Total no. of collisions	43
Hit Ratio	60.5%

Table 1: Results from NTSB accident reports.

the linkages returned from Link Grammar for these sentences are either incomplete or incorrect. The accident report summaries are written in bureaucratic American English with sometimes very long sentences with many subordinate clauses which make it very hard for Link Grammar to find complete linkages.

4 Related Work

Mokhtar and Chanod [AMC] uses another approach for extracting subject-object relationships. They use an incremental finite-state parser to annotate the input string with syntactic markings, and them pass the annotated string through sequence of finite-state transducers. Their results are much better than our experience with Link Grammar with a precision $\approx 90\%$ for both subjects and objects.

Ferro et al. [FVY99] uses a trainable error-driven approach to find grammatical relationships. The achieved precision running on a test set is $\approx 77\%$.

Brants et al. [BSK97] have a slightly different goal with their research. They have developed an interactive semi-automatic tool for constructing treebanks. Their results are very good, $> 90\%$, but then one has to have in mind that the only automatic annotation is to assign grammatical function and/or phrase category.

5 Conclusions and Future Work

We have shown that using Link Grammar together with regular expression pattern matching is a plausible technique for extracting collisions from, also grammatically complicated, texts. Achieving a hit ratio of 60.5% on the NTSB report summaries is a good result considering both the simplicity of the collision detector and the grammatical complexity of the tested sentences.

Since most of the collision detection problems originate in incomplete and/or incorrect linkages returned from Link Grammar, this is the obvious first choice for possible improvements. Adding domain-specific knowledge to Link Grammar so that it would concentrate on linking collision verbs with their subject and object instead of trying to create a complete linkage over the complete sentence would significantly enhance the hit ratio of the collision detector.

Other improvements to the collision detector to further enhance the hit ration somewhat include extending the regular expression patterns to match even more complicated linkages than is possible in the current implementation.

References

[AMC] Salah Aït-Mokhtar and Jean-Pierre Chanod. Subject and object dependency extraction using finite-state transducers. Rank Xerox Research Centre, Meylan, France.

[BSK97] Thorsten Brants, Wojciech Skut, and Brigitte Krenn. Tagging grammatical functions. In *Proceedings of EMNLP-2*, July 1997.

[DELN01] Sylvain Dupuy, Arjan Egges, Vincent Legendre, and Pierre Nugues. Generating a 3d simulation of a car accident from a written description in natural language: The CarSim system. In *Proceedings of The Workshop on Temporal and Spatial Information Processing*, pages 1–8. ACL, July 2001.

[Fel] Christiane Fellbaum. English verbs as a semantic net. *http://www.cogsci.princeton.edu/ wn/*.

[FVY99] Lisa Ferro, Marc Vilain, and Alexander Yeh. Learning transformation rules to find grammatical relations. In *Computational Natural Language Learning*, pages 43–52. ACL, June 1999.

[NTS] The national traffic safety board. *http://www.ntsb.gov*.

[ST] Daniel D. Sleator and Davy Temperly. Parsing english with a link grammar. *http://www.link.cs.cmu.edu/link/*.

A Collision Verbs

This list of collision verbs in collected from WordNet. The WordNet Browser was used to find synonyms to the verb *strike* in its collide sense. That procedure was recursively repeated to find all appropriate collision verbs.

collide, clash To crash together with violent impact. *The cars collided. Two meteors clashed.*

crash, ram To undergo damage or destruction on impact. *The plane crashed into the ocean. The car crashed into the lamp post.*

hit, strike, impinge on, run into, collide with To hit against or come into sudden contact with something. *The car hit a tree. He struck the table with his elbow.*

rear-end To collide with the rear end of something. *The car rear-ended me.*

broadside To collide with the broad side of something. *Her car broad-sided mine.*

bump, knock To knock against something with force or violence. *My car bumped into the tree.*

run into, bump into, jar against, butt against, knock against To collide violently with an obstacle. *I ran into the telephone pole.*

B Tested Sentences

HAR0001 As the bus approached milepost (MP) 184.9, it traveled off the right side of the roadway into an "emergency parking area," where *it* **struck** *the back of a parked tractor-semitrailer*, which was pushed forward and struck the left side of another parked tractor-semitrailer.

Result:

1. it struck the back of a parked tractor-semitrailer
2. a parked tractor-semitrailer struck the left side of another parked tractor-semitrailer

Comment Both collisions correctly found. Co-reference of collision 1 not resolved.

HAR0002 As *the bus* approached the intersection, *it* failed to stop as required and **was struck by** *the dump truck*.

Result:

1. the bus was struck by the dump truck

Comment Collision correctly found.

HAR0101 *The bus* continued on the side slope, **struck** *the terminal end of a guardrail*, traveled through a chain-link fence, vaulted over a paved golf cart path, **collided with** *the far side of a dirt embankment*, and then bounced and slid forward upright to its final resting position.

Result:

1. *Collision not found*
2. a paved golf cart path collided with the far side of a dirt embankment

Comment Subject and object not founf at all in collision 1, and incorrect subject in collision 2. Both due to incorrect LinkGrammar linkages.

HAR0102 About 90 seconds later, *northbound Metrolink commuter train 901*, operated by the Southern California Regional Rail Authority, **collided** with *the vehicle*.

Result:

1. 901 collided with the vehicle

Comment Collision found, but subject not fully resolved due to incomplete linkage.

HAR0103 Abstract: On March 28, 2000, about 6:40 a.m. (sunrise was at 6:33 a.m.), *a CSX Transportation, Inc., freight train* traveling 51 mph **struck** *the passenger side of a Murray County, Georgia, School District school bus* at a railroad/highway grade crossing near Conasauga, Tennessee.

Result:

1. train traveling struck the passenger side of a County

Comment: Collision correctly found, though object is not completely resolved. Works with proper name substitution.

HAR9001 Comment: No collisions in this report.

HAR9002 About 7:34 a.m ., central daylight time, on Thursday, September 21, 1989, *a westbound school bus* with 81 students operated by the Mission Consolidated Independent School District, Mission, Texas, and *a northbound delivery truck* operated by the Valley Coca-Cola Bottling Company, McAllen, Texas, **collided** at Bryan Road and Farm to Market Road Number 676 (FM 676) in Alton, Texas.

Result:

1. Company collided Number

Comment: Incorrect subject and object due to erroneous linkage.

HAR9003 Comment: No collisions in this report.

HAR9101 About 5:40 p.m. on July 26,1990, *a truck* operated by Double B Auto Sales, Inc., transporting eight automobiles entered a highway work zone near Sutton, West Virginia, on northbound Interstate Highway 79 and **struck** *the rear of a utility trailer* being towed by a Dodge Aspen.

The Aspen then **struck** *the rear of a Plymouth Colt*, and *the Double B truck* and *the two automobiles* traveled into the closed right lane and **collided with** *three West Virginia Department of Transportation (WVDOT) maintenance vehicles.*

Result:

1. 79 struck the rear of a utility trailer
2. the Aspen struck the rear of a Colt
3. the B truck collided with three

Comment: Incorrect subject in collision 1. Collision 2 is correct. In collision 3 there are two subjects of which only the first is found.The incorrect object in collision 3 is correct when using proper name substitution.

HAR9201 Comment: No collisions in this report.

HAR9202 About 9: 10 a.m. on December 11, 1990, *a tractor-semitrailer* in the southbound lanes of 1-75 near Calhoun, Tennessee, **struck** *the rear of another tractor-semitrailer* that had slowed because of fog.

After the initial collision, *an automobile* **struck** *the rear of the second truck* and was in turn **struck** in the rear **by** *another tractor-semitrailer.*

Meanwhile, in the northbound lanes of 1-75, *an automobile* **struck** *the rear of another automobile* that had slowed because of fog.

Result:

1. *Collision not found.*
2. an automobile struck the rear of the truck

3. *Collision not found.*
4. an automobile struck the rear of another automobile

Comment: Collision 1 is found when using proper name substitution. Collision 3 is represented by a complicated linkage which is not properly dealt with. Collision 2 and 4 are correctly found though the object of collision 2 is not completely resolved.

HAR9301 During the descent, *the bus* increased speed, left the road, plunged down an embankment, and **collided with** *several large boulders*.

Result:

1. the bus collided with several large boulders

Comment: Correctly found collision.

HAR9302 *The bus* **struck** *a car*, overturned on its right side, slid and spun on its side, uprighted, and **struck** *another car* before coming to rest.

Result:

1. the bus struck a car
2. the bus struck another car

Comment: Both collisions correctly found.

HAR9401 About 3:13 p.m., Wednesday, March 17, 1993, *an Amerada Hess (Hess) tractor-semitrailer* hauling gasoline **was struck by** *National Railroad Passenger Corporation (Amtrak) train 91*.

The truck, which was loaded with 8,500 gallons of gasoline, was punctured when *it* **was struck**.

Result:

1. an Amerada tractor-semitrailer struck by Corporation
2. *Collision not found.*

Comment: Incorrect linkage in Collision 1. Collision 2 is no collision and is therefore not found though a collision verb occurs.

HAR9402 On May 19, 1993, at 1:35 a.m., while traveling south on Interstate 65 near Evergreen, Alabama, *a tractor* with bulk-cement-tank semitrailer left the paved road, traveled along the embankment, overran a guardrail, and **collided with** *a supporting bridge column* of the County Road 22 overpass.

An automobile and *a tractor-semitrailer*, also southbound, then **collided with** *the collapsed bridge spans*.

Contributing to the severity of the accident was the collapse of the bridge, after *the semitrailer* **collided with** and demolished *the north column*, that was a combined result of the nonredundant bridge design, the close proximity of the column bent to the road, and the lack of protection for the column bent from high-speed heavy-vehicle collision.

Result:

1. a tractor collided with a supporting bridge column of the County 22 overpass
2. an automobile collided with the collapsed bridge spans
3. *Collision not found.*
4. *Collision not found.*

Comment: Incorrect linkages in collisions 3 and 4.

HAR9403 About 3:30 p.m. CDT on May 28, 1993, *the towboat CHRIS*, pushing the empty hopper barge DM 3021, **collided with** *a support pier of the eastern span of the Judge William Seeber Bridge* in New Orleans, Louisiana.

Result:

1. CHRIS collided with a support pier of the Bridge

Comment: Collision found correctly.

HAR9501 Seconds later, *National Railroad Passenger Corporation (Amtrak) train number 88*, the Silver Meteor, carrying 89 passengers, **struck** *the side of the cargo deck and the turbine.*

Result:

1. Meteor struck the side of the cargo deck

Comment: Collision found, but the subject can not be resolved due to erroneous linkages.

HAR9502 *The truck* drifted across the left lane onto the left shoulder and **struck** *the guardrail*; *the tank* **hit** *a column of the Grant Avenue overpass.*

Result:

1. the truck struck the guardrail
2. the tank hit a column of the Avenue overpass

Comment: Both collisions found and resolved.

HAR9503 As *the lead vehicle* reportedly slowed from 65 miles per hour (mph) to between 35 and 40 mph, *it* **was struck** *in the rear.*

Subsequent collisions occurred as *vehicles* **drove into** *the wreckage area* at speeds varying from 15 to 60 mph.

Result:

1. the lead vehicle was struck the rear
2. *Collision not found.*

Comment: Collision 1 found and almost completely resolved. Collision 2 could not be found due to complicated (erroneous?) linkages.

HAR9601 About 35 minutes later, *the truck* **was struck by** *southbound Amtrak train No. 81* en route from New York City to Tampa, Florida.

Result:

1. the truck was struck by southbound Amtrak

Comment: Collision found, object could not be fully resolved.

HAR9602
Comment: No collisions in this report.

HAR9701S Abstract: On November 26, 1996, *a utility truck* **collided with** and fatally injured *a 10-year-old student* near Cosmopolis, Washington.

Result:

1. *Collision not found.*

Comment: Incorrect linkage.

HAR9702S Abstract: On April 25, 1996, *a truck* with a concrete mixer body, unable to stop, proceeded through an intersection and **collided with** and overrode *a passenger car* near Plymouth Meeting, Pennsylvania.

Result:

1. *Collision not found.*

Comment: Incorrect linkage.

HAR9801S Abstract: On June 11, 1997, *a transit bus* **collided with** *seven pedestrians* at a "park and ride" transit facility in Normandy, Missouri.

Result:

1. a transit bus collided with seven pedestrians

Comment: Correct.

HAR9801 *A flatbed truck* loaded with lumber, operated by McFaul Transport, Inc., that was traveling southbound on U.S. Route 41 **collided with** *the doubles truck*, lost control, and crossed over the median into the northbound lanes.

A northbound passenger van with nine adult occupants **struck** and underrode *the right front side of the flatbed truck* at the landing gear.

A refrigerator truck loaded with produce, operated by Glandt/Dahlke, Inc., *that* was also traveling northbound, **struck** *the right rear side of the flatbed truck.*

Result:

1. 41 collided with the doubles truck
2. *Collision not found.*
3. a refrigerator truck struck the right rear side of the flatbed truck

Comment: Collision 2 not found because of erroneous linkage.

HAR9802S Abstract: On October 9, 1997, about 12:10 a.m., *a truck tractor* pulling a cargo tank semitrailer was going under an overpass of the New York State Thruway when *it* **was struck by** *a sedan*.

The car **hit** *the right side of the cargo tank* in the area of the tank's external loading/unloading lines, releasing the gasoline they contained.

Result:

1. it was struck by a sedan
2. the car hit the right side of the cargo tank

Comment: Coreference in collision 1 not found, otherwise correct.

HZM9101 *The vehicle* overturned onto its side and **struck** *the embankment of a drainage ditch* located in a dirt field beside the road.

Result:

1. the vehicle struck the embankment of a drainage ditch

Comment: Correct.

HZM9901

Comment No collisions in this report.

RAR0201 Executive Summary: About 9:47 p.m. on March 15, 1999, *National Railroad Passenger Corporation (Amtrak) train 59*, with 207 passengers and 21 Amtrak or other railroad employees on board and operating on Illinois Central Railroad (IC) main line tracks, **struck** and destroyed *the loaded trailer of a tractor-semitrailer combination* that was traversing the McKnight Road grade crossing in Bourbonnais, Illinois.

The derailed Amtrak cars **struck** *2 of 10 freight cars* that were standing on an adjacent siding.

Result:

1. *Collision not found*
2. the derailed Amtrak cars struck 2 of 10 freight cars

Comment: Collision 1 not found because of erroneous linkage.

RHR9001 About 9:38 a.m., Pacific standard time, on December 19, 1989, *National Railroad Passenger Corporation (Amtrak) passenger train 708*, consisting of one locomotive unit and five passenger cars, **struck** *a TAB Warehouse & Distribution Company tractor semitrailer* in a dense fog at a highway grade crossing near Stockton, California.

Result:

1. *Collision not found.*

Comment: Incorrect linkage.

Document Classification for Computer Science Related Articles

May 15, 2002

Ana Fuentes Martínez
Flavius Gruian

Introduction to Natural Language Processing and Computational Linguistics

Document classification based on content is a general task that appears in a wide spectrum of tasks from text retrieval to automatic browsing tools or database maintenance. It is expensive when performed manually –even though this is still the most accurate method. In this project we have implemented a simple probabilistic document classifier based on corpus similarity from a predefined topic hierarchy. It is based on the Naïve Bayes method with multinomial sampling well suited for related text classes. The method was evaluated with a set of articles from three different journals in Computer Science. The experiments showed that Naïve Bayes classifier was able to correctly determine the class of 93% of the abstracts in the test set, for well balanced training data.

1. Introduction

The question of whether an automatic classifier performs better than humans is a longstanding controversy. Certainly it depends on how skilled the person is, but also on the amount of documents to be classified. Generic classifiers often have poorer performance than the ones using domain-specific methods [1] but humans taggers will also find greater difficulties in categorizing wide-ranging text sets. Time and cost effectiveness issues have contributed in making automatic tools an attractive alternative and promoted research in the field.

Several machine learning techniques have been developed to automatically discern text by content. Indexing non-homogeneous documents, as the ones found on the World Wide Web is discussed in a recent survey by Y.Yang et. al. [2] It focuses on well-known algorithms (Naïve Bayes, Nearest Neighbor and First Order Inductive Learner) applied to hypertext categorization. The sparseness problem caused by large number of categories with few documents on each is examined in [3]. Estimates of the terms distributions are made by differentiation of words in the hierarchy according to their level of generality/specificity. Some practical work has been done within Web page clustering. We mention two different approaches to organize a set of documents. In [4] categories are being constructed from a set of unclassified texts. The documents are clustered together according to semantic similarity criteria. This is certainly the most frequent practice in text classification. A different idea, syntactic clustering, is discussed in [5]. Their intention is to determine whether two documents are “roughly the same” or “roughly contained” except from modifications such as formatting or minor corrections.

More homogeneous texts are often indexed in predefined categories. They are characterized by containing multiple overlapping terms across the classes. One example of this type of corpus is presented in [7]. Machine learning techniques are used to detect whether or not literary works can be classified according to the genders of their authors just by noting differences in the type of prose. The subject of the classifier proposed in [8] are bioinformatics articles. The general idea behind this work is that the features useful for distinguishing articles vary among the different levels in the hierarchy.

Classifiers can be used for either category ranking or automated category assignments (binary classification). Both require the same kind of computation and deduce the same information [9]. The type of output depends on the application. For user interaction a category ranking system would be more useful while for fully automated classification tasks category assignment is desirable.

2. Categorization Model

a) Naïve Bayes probabilistic classifier

We assume that a predefined set of text categories is given. A large set of abstracts from each of the categories conform the corpus. Each document is presumed to belong to a single category.

Based on this corpus we define a probability model for abstracts. New documents are classified following Bayes' rule by computing posterior probabilities over the categories. Naïve Bayes probabilistic classifiers are commonly used in text categorization. The basic idea is to use the joint probabilities of words and categories to estimate the probabilities of categories given a document [10]. It is easy to estimate the probability of each term by simply counting the frequency with which the target value occurs in the training data:

$$P(w_i|Class) = \frac{Occurrences\ of\ w_i\ in\ Class}{No.\ of\ Words\ in\ the\ Class(C)} \quad (1)$$

The naive part of such an approach is the assumption of word independence [10]. In other words, the probability of observing the conjunction $w_1,w_2,...,w_n$, is just the product of the probabilities for the individual attributes.

$$P(w_1, w_2, \ldots w_n|C_j) = \prod_i P(w_i|C_j) \quad (2)$$

Notice that in a Naïve Bayes learner, the number of distinct $P(w_i|C_i)$ that must be calculated is significantly smaller than all possible $P(w_1,w_2,...,w_n,|C_i)$. Despite the independence assumption is clearly incorrect –it is more probable to find the word *network* if the preceding word is *neural*– the algorithm performs well in many text classification problems.

The document *D* is assigned to the class that returned greatest probability of observing the words that were actually found in the document, according to the formula

$$Class = \arg\left[max_{C_j \in C}\left\{ P(C_j) \cdot \prod_{w_i \in D} P(w_i|C_j) \right\}\right] \quad (3)$$

b) Canonical form of the document

The first main issue involved in applying the Naïve Bayes classifier to text classification problems is to decide how to represent an arbitrary text document in terms of attribute values. Beside word independence, we made an additional reasonable assumption regarding the distribution of terms in a document; the probability of encountering a specific word in a document is independent of its position. The primary advantage is that it increases the number of examples available for each term and therefore improves the reliability of the estimates.

A Prolog written filter parses the document and produces what we call the canonical form of a text. This canonical form is actually a vector representation of the terms and their frequencies. In order to reduce the amount of data, the canonical form contains only the information that is relevant for the probabilistic measures. Punctuation marks, numbers and stopwords are removed and uppercase letters are converted to lowercase so that terms can be compared. Notice that this approach implies strictly semantic similarity, no syntactic issues are considered.

c) Bayesian Estimation with Uniform Prior

To complete the design of our leaning model we must choose a method for estimating the probability terms required by the Naïve Bayes classifier. The word frequencies can be obtained by a Bayesian Estimation with Uniform Priors, also known as Laplace's smoothing law [6]. It addresses the problem of assigning a probability distinct from zero to those terms in the document that do not appear in the corpus. It can be interpreted as enlarging the corpus with the current text. For the formula (1) we have that the size of the new vocabulary is the number of words in the class (or size of the class), *C* plus the size of the document, *N*. The word counter for w_i is increased by one because now w_i is present also in the document.

$$P(w_i|C_j) = \frac{|w_i| + 1}{C + N} \qquad (4)$$

d) The system

The system we designed has two major parts: A prolog program for parsing the new document and a Perl script for computing the probabilities and selecting the most suitable category. Evaluation of the performance was done automatically by iterating over a directory of test files and checking whether the class assignment suggested by the algorithm matched the original journal, which was apparent from the file name.

During the leaning stage, the algorithm examines all training documents to extract the vocabulary for each class and count the frequency of all terms. These corpora are also presented in the canonical form.

Later, given a new document to be classified, the probability estimates from each corpus are used to calculate the closest category according to the formula (3). The result is obtained as a ranked list of the candidate categories where the first one is consider to be correct and the others are disregarded.

3. Results and Discussion

a) Experimental set-up

The corpora used to run the experiments consisted of a large set of abstracts from computer science journals. They were divided into thee clusters according to their main topic: Digital Signal Processing, Biomedical Research and Computer Speech & Language, all collected from Academic Press Journals. The final training data contained over 7000 different significant terms i.e, without stopwords or numbers. The corpora enclosed more than 21.000 words. The classifier was tested with 60 new abstracts from the same journals. This way the correct class was known in advance and it could be compared to the one suggested by the classifier. The division between the training set and the test set was randomly chosen and several configurations were tested in order to ensure reliable results.

Two characteristics differentiate our data set compared to other papers in the field. First, we are working with rather coarse data. The different classes will include larger amount of overlapping terms which will blur the fitting measures. This is not the case for general web classifiers with rather heterogeneous document topics. Second, it is typical for technical articles to contain many very specific terms –and this is specially outstanding in the abstract. A hand-tuned classifier could make use of this information to increase the weight of this terms in the probabilities and achieve better classification rates. Each article is assigned a single category in the hierarchy.

b) Evaluation Measures

We used a four cell contingency table for each category and a set of test documents:

Table I: contingency table for a class

	documents that belong to C	documents that DO NOT belong to C
documents assigned to C	a	b
documents NOT assigned to C	c	d

The total number of test documents is

$$N = a + b + c + d$$

where

- a is the number of documents correctly assigned to this category.
- b is the number of documents incorrectly assigned to this category.
- c is the number of documents incorrectly rejected from this category.
- d is the number of documents correctly rejected from this category.

We obtained the following results:

Table II: contingency table for Digital Signal Processing

	documents that belong to Digital Signal Processing	documents that DO NOT belong to Digital Signal Processing
documents assigned to Digital Signal Processing	17	0
documents NOT assigned to Digital Signal Processing	3	40

Table III: contingency table for Biomedical Research

	documents that belong to Biomedical Research	documents that DO NOT belong to Biomedical Research
documents assigned to Biomedical Research	19	0
documents NOT assigned to Biomedical Research	1	40

Table IV: contingency table for Computer Speech & Language

	documents that belong to Computer Speech & Language	documents that DO NOT belong to Computer Speech & Language
documents assigned to Computer Speech & Language	20	4
documents NOT assigned to Computer Speech & Language	0	36

Conventional performance measures are defined and computed from this contingency tables. This measures are recall, *r*, precision, *p*, fallout, *f*, accuracy, *Acc*, and error, *Err*:

- $r = a / (a + c)$, if $a + c > 0$, otherwise undefined;
- $p = a / (a + b)$, if $a + b > 0$, otherwise undefined;
- $f = b / (b + d)$, if $b + d > 0$, otherwise undefined;
- $Acc = (a + d) / N$;
- $Err = (b + c) / N$.

Table V: Performance measures

	recall	precision	fallout	accuracy	error
Digital Signal Processing	85%	100%	0%	95%	5%
Biomedical Research	95%	100%	0%	98%	2%
Computer Speech & Language	100%	83%	10%	93%	7%

Macro and Micro-averaging

For evaluating performance of the system across categories, there are two conventional methods, namely macro-averaging and micro-averaging. Macro-averaged performance scores are calculated by first computing the scores for the contingency tables for each category and then averaging those to obtain global means. This is a per-category average, which gives equal weight to every category regardless of its frequency.

Table VI: Macro-Averaged Performance Measures

	recall	precision	fallout	accuracy	error
Macro-Averaging	93,3%	94,3%	3,3%	95%	4,6%

Micro-averaged performance scores give equal weight to every document. this can be more representative in our case, since the number of documents is small and all the classes were tested with the same number of abstracts. First we create a global contingency table, with the average of the values in the category tables, and use this to infer the performance scores.

Table VII:global contingency table for a class

	documents that belong to C	documents that DO NOT belong to C
documents assigned to C	17+19+20 = 56	0+0+4 = 4
documents NOT assigned to C	3+1+0 = 4	40+40+36 = 116

Table VIII:Micro-Averaged Performance Measures

	recall	precision	fallout	accuracy	error
Micro-Averaging	93%	93%	3%	96%	4%

c) Unbalanced Corpora.

Some poor results – bellow 30% successfully classified documents– were obtained in the first versions of the program. Two circumstances were pointed out as possible reasons: The corpus was both small and unbalanced.

The abstracts in one of the journals tended to be significantly shorter than the average so both the corpus and the test documents from this class contained less information. Therefore, very few abstracts were finally assigned to this cluster.

Enlarging the corpora proved to be a better solution than artificially balancing the class corpus. After having collected a corpus of near 20000 words, the classification rate reached 93% correctly assigned documents, still with a rather unbalanced distribution (Coefficient of Variance = 20%). Neither larger corpora nor better distribution resulted in reduced misclassification (see figure 1).

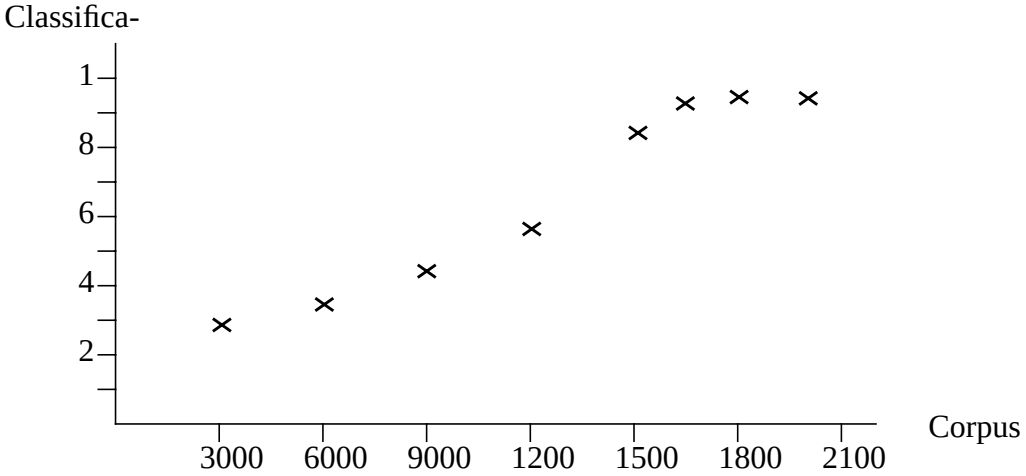

Figure 1. Classification rates with respect to the size of the corpus. Coefficient of Variance = 0,2

Above this threshold, enlarging the corpora had little effect on the number of new significant terms for each class and consequently on the performance (see figure 2). This suggests that the presence/absence of a common term is clearly more revealing for a good classification than the number of occurrences. Because of this property, words such as *model* or *system* that appear often in all classes will have restricted effect in the classification decision compared to those terms which are unique for a class.

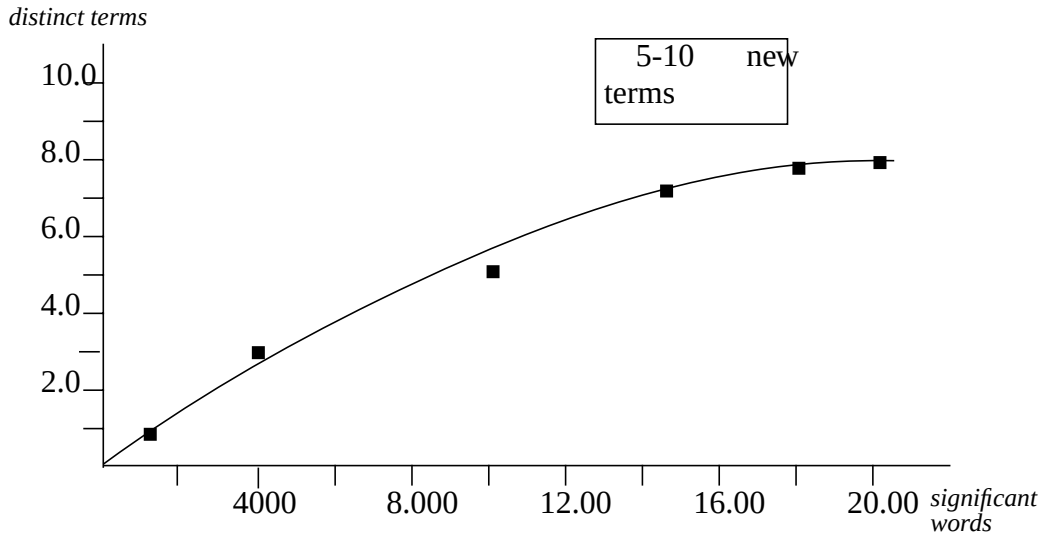

Figure 2. Relation between the size of the corpus and the number of different terms on it.

References

[1] W. Cohen. *Learning to classify English text with ILP methods*. In luc De Raedt, editor. Advances in Inductive Logic Programming, p. 124-143 IOS Press 1995.

[2] Y Yang, S. Slattery and R. Ghani. *A study of approaches to hypertext categorization.* Journal of Intelligent Information Systems, Volume 18, Number 2, March 2002.

[3] Kristina Toutanova, Francine Chen, Kris Popat & Thomas Hofmann. *Text Classification in a Hierarchical Mixture Model for Small Training Sets*, Xerox PARC and Brown University, Proceedings 10th International Conference on Information and Knowledge Management, 2001.

[4] Paul Ruhlen, Husrev Tolga Ilhan, and Vladimir Livshits *Unsupervised Web Page Clustering* CS224N/Ling237 Final Projects Spring 2000, Stanford Univ.

[5] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig. *Syntactic clustering of the Web.* In Proc. 6th WWW Conf., Apr. 1997.

[6] Sep Kamvar & Carla Pinon, *Probabilistic Smoothing Models for Automatic Text Classification* CS224N/Ling237 Final Projects Spring 2001, Stanford University.

[7] Zoe Abrams, Mark Chavira, and Dik Kin Wong, *Gender Classification of Literary Works* CS224N/Ling237 Final Projects 2000, Stanford University.

[8] Jeff Chang, *Using the MeSH Hierarchy to Index Bioinformatics Articles* CS224N/ Ling237 Final Projects 2000, Stanford University.

[9] Yiming Yang *An evaluation of statistical approaches to text categorization.* Journal of Information Retrieval, Vol 1, No. 1/2, pp 67--88, 1999.

[10]Tom M. Mitchell. Machine Learning. McGraw Hill, 1997.

[11]http://www.ncbi.nlm.nih.gov/entrez/query/static/help/pmhelp.html#stopwords

Anagram generation using sentence probabilities

Sven Gestegård Robertz
Department of Computer Science, Lund University
email: sven@cs.lth.se

June 3, 2002

Abstract

An anagram is a word or phrase made by transposing the letters of another word or phrase, and is a popular kind of puzzle. Finding anagrams for a word or a phrase is a difficult task due to the huge number of possible permutations of the letters, and using computers for anagram generation has been increasingly popular. This paper presents an attempt on constructing an automatic anagram generator based on sentence probabilities. Using a relatively small corpus of one million words, a simple implementation using bigram probabilities proved to yield quite satisfactory results but is too inclusive due to the limited context recorded in bigrams. A trigram based version produces much less incorrect results but would need a larger corpus since it is much more exclusive.

1 Introduction

An anagram[1] is a word or phrase made by transposing the letters of another word or phrase and are popular as puzzles. It is also common to make humorous and/or abusive anagrams of names of people and places. Anagrams have also been believed to have mystical or prophetic meaning, especially when applied to religious texts.

Finding anagrams on long words or sentences is very difficult because of the huge number of possible permutations of the letters. Therefore, it is desirable to use computers to search for anagrams. Currently, there exists a large number of anagram generation programs[7]. Many are available on the world wide web both for download and as online services[5, 2]. The common approach is to use a dictionary to limit the amount of possible anagrams to those consisting of actual words. This method yields a huge number of

anagrams, and the automatically generated result must be filtered manually in order to find the ones that actually mean something.

The main problem with automatic anagram finders is to select the anagrams that actually have a meaning (in a certain language) from the huge number of possible permutations of the letters. If we limit ourselves to single-word anagrams the problem is easily solved by dictionary lookup, but if we also want to find multi-word anagrams, the problem not only to find proper words but also to select the series of words that actually have a meaning.

The idea in this work is to reduce the number of false hits by using a statistical method to select the only the most probable sentences.

2 Statistical methods

A *corpus*[9, 4, 10, 8] is a large collection of texts or speech. Corpora are used for statistical analysis of natural language. Two application areas are lexicography[3] and language recognition. In lexicography, corpora are used to find the most common uses of a word in order to select proper examples and citations. In speech and optical character recognition, statistics are used to select the most probable among a set of similarly sounding or looking words given the previous words.

While grammar based methods require detailed knowledge and careful hand-coding, statistical methods are much more flexible and are easy to adapt to different styles of writing (e.g., a scientific corpus and a newspaper corpus in the same language would be quite different) or to different languages, especially with the huge amounts of electronically available text, on the world wide web and electronic libraries.

2.1 Sentence probabilities

For a sentence $S = w_1 w_2 w_3 \dots w_n$ the sentence probability is

$$P(w_1 w_2 \dots w_n) = P(w_1) \times P(w_2|w_1) \times P(w_3|w_1 w_2) \times P(w_n|w_1 \dots w_{n-1})$$

It is, however, not practically possible to use this definition of sentence probability for actual calculations, since it would require an infinitely large corpus in order to accept all correct sentences in a language. Instead, the sentence probability has to be approximated in some way.

2.2 n-gram probabilities

The sentence probability can be approximated by using n-grams, i.e., the conditional probability of a word given the $n-1$ previous words. In this work, we use bigrams and trigrams. The bigram probability of a sentence is

$$P_{bigram}(S) \approx P(w_1) \times P(w_2|w_1) \times \ldots \times P(w_n|w_{n-1})$$

where the probability of a bigram $w_k w_{k+1}$ is calculated as

$$P(w_k w_{k+1}) = P(w_k|w_{k+1}) = \frac{freq(w_k w_{k+1})}{freq(w_k)}$$

The trigram probability is

$$P_{trigram}(S) \approx P(w_1) \times P(w_2|w_1) \times P(w_3|w_1 w_2) \ldots \times P(w_n|w_{n-2} w_{n-1})$$

Using unigram probabilities is equal to dictionary lookup, i.e., all permutations of the words are displayed. The threshold probability effectively sets the size of the dictionary.

2.3 Drawbacks of relying on statistics

The drawbacks of relying entirely on statistics is that it does not take any (formal) semantical knowledge about the language into account. For instance, a certain compound noun may not be very probable in a given corpus, but is a valid anagram anyhow. And a series of words, that are very probable in the corpus, may not be a full sentence and should not be presented as an anagram since they, in isolation, have no meaning.

Thus, relying solely on statistics, without any formal semantic analysis, will yield a too limited system. Such a system will be too inclusive in some respects and too exclusive in others.

3 Implementation

The basic algorithm for generating anagrams is as follows

- The algorithm keeps three strings:

  prev is the start of the anagram that has been accepted as probable,

  head is the start of the word(s) currently being built, and

  rem is the remaining, not yet used, letters.

- for each letter in *rem*, add it to *head*.
  - If it produces a valid word (and *prev* + *head* is a probable series of words), append *head* to *prev* and find recursively find the set of words that can be built from the remaining letters of *rem*.
  - Continue recursively until all letters have been used.

A common way to reduce the number of branches in the recursion tree is to use signatures (the set of characters in a string, rather than the string itself) in the recursive search and then look up all words that match (i.e., can be built from) a certain signature on the way up. I also try to cut of branches that are dead ends early by stopping the search if the `prev` string doesn't contain a probable series of words.

As an example, consider the phrase "think different". In some stage of execution, the algorithm will be in the following state: a word ("fifteenth") has been found, and the search continues, recursively on the remaining letters.

```
(prev,head,rem) = ("fifteenth","", "dikrn")
```

Then we select one letter from *rem* and continue the search recursively. (Since there is only one vowel in *rem*, there is effectively only one path and it isn't interesting to try and build more than one word from the letters of *rem*.)

```
("fifteenth","d", "ikrn")
-->
("fifteenth","di", "krn")
-->
...
-->
("fifteenth","dikrn", "")
```

Now, we look in our dictionary to find any words that can be built from the letters "dikrn". We find the word "drink" and add that to *prev*.

```
(prev,head,rem) = ("fifteenth drink","", "")
```

All the letters have been used, so the recursion stops and we return the result.

Figure 1 shows the algorithm i a little more detail using Java-like pseudo code. The search is done recursively in the method `searchRecursive(prev, head, rem)`.

```
Collection searchRecursive(String prev, String head, String rem){
/* returns collection of strings */

    Collection result;

    if(rem.length() == 1) {
        String candidate = head+rem;

        if(sp.sigExists(candidate)) {
            result = sp.getWordsForSig(candidate);
         } else {
            result = null;
        }
    } else {
        if(hasProbableSentences(prev) ) {
            result = doActualSearchExpand(prev, head, rem, depth);
            result = getProbableSentences(prev,result);
        } else {
          result = null;
        }
   }
    return result;
}

Collection doActualSearchExpand(String prev, String head, String rem){

    Collection result;
    for( each character c in rem ) {
        String newRem = rem - c;
        String newHead = head + c;
        if(hasVowels(newHead)) {
            heads = sp.getWordsForSig(newHead);
        }
        if(heads > 0 && hasVowels(newRem)) {
            String newPrev = concat(prev, newHead);
            theRest = searchRecursive(newPrev, "", newRem);

            result.addAll( allCombinations(heads, theRest));
        }
        tmp = searchRecursive(prev, head+c, s.toString(), depth+1);
            result.addAll(tmp);
        }
   }
   return result;
}

Collection allCombinations(Collection first, collection second){
    return all combinations (x,y) where x in first and y in second
}

boolean sigExists(String signature) {
   return true if any word in the corpus matches the signature, else false
}

Collection getProbableSentences(String head, collection tails){
    Collection result;
    for( each String t in tails) {
        candidate = head + " " +  t;
        if(getSentenceProbability(candidate) > THRESHOLD) {
            result.add(candidate);
        }
    }
    return result;
}
```

Figure 1: The anagram generation algorithm.

4 Experimental results

This section presents the results of sample executions of the program using a Swedish and an English corpus, respectively. The results illustrate the difference between using bigram and trigram probabilities; using trigram probabilities is much more exclusive .

The Swedish corpus was the complete works of Selma Lagerlf (962497 words), and the word to be anagrammed was "universitet". In this case, using bigrams gave 88 anagrams whereas using trigrams only produced one.

The English corpus was randomly selected from the Project Gutenberg electronic library[6] and consisted of Rudyard Kipling, *The Works*; Julian Hawthorne, *The Lock and Key Library/Real Life*; Bennett, *How to Live on 24 Hours a Day*; and O Henry, *Waifs and Strays, etc.* The size was 723910 words (including the Project Gutenberg copyright notice). The phrase to be anagrammed was "anagram generator".

For each anagram, the sentence probability is given (the limit was 10^{-10}).

4.1 Results from using bigram probabilities

```
Anagrams for "universitet":
en ser ut i vit [p=1.9458480510721317E-10]   ser ut i vinet  [p=1.015513774753907E-6]
envis i ett ur  [p=5.142381068564074E-7]     ser ut inte vi  [p=2.9621187688537784E-8]
envis i tu tre  [p=4.440915397008955E-7]     ser ut vi inte  [p=3.0729917387234457E-6]
envist i er ut  [p=1.7160445913500644E-7]    ser vi i ett nu [p=1.8823062145991216E-10]
er nu vi sett i [p=1.937940900390817E-10]    ser vi inte ut  [p=3.7808359616827964E-7]
er vi nu sett i [p=7.308805681473939E-10]    ser vi nu i ett [p=1.1218191968677428E-8]
i ett ur sven i [p=1.5516368737656366E-10]   ser vi ut inte  [p=2.7759636302831487E-9]
ii ett ur sven  [p=3.0035429642629304E-8]    sett i er nu vi [p=2.452672726387112E-10]
in reste ut vi  [p=5.891550112985203E-9]     sitter i nu ve  [p=1.7904648728424079E-9]
in vi reste ut  [p=1.58599764968365E-8]      sven i tu tre i [p=5.816617555780169E-10]
inte ser ut vi  [p=9.478780060332091E-8]     tu tre i en vis [p=7.664376858367229E-10]
inte ut vi ser  [p=5.902352680503676E-9]     tu tre i sven i [p=5.816617555780167E-10]
ner se ut i vit [p=3.96659435500385E-10]     tvi ni se er ut [p=1.4087963583645376E-9]
ni reste ut vi  [p=2.4219565474030946E-8]    tvi ni se ut er [p=1.1592270654237034E-8]
ni ut vi reste  [p=8.126785388846779E-10]    ur sven i ett i [p=2.327455310648455E-10]
ni vi reste ut  [p=1.0866459358890525E-8]    ut i sven i tre [p=1.1661532621133962E-10]
nu i er vi sett [p=1.3060234777380274E-10]   ut inte ser vi  [p=3.4678463635361306E-9]
nu reste i vit  [p=2.0237228371499173E-9]    ut inte vi ser  [p=9.222426063286992E-10]
nu ser vi i ett [p=1.3830647632616007E-9]    ut se er tvi ni [p=1.267722806993553E-10]
nu ve er i sitt [p=9.040149103118745E-10]    ut se i vinter  [p=3.998242714219306E-10]
nu vet i riset  [p=5.242222049351395E-10]    ut vi in reste  [p=5.930663383767719E-10]
nu vi ser i ett [p=1.8627134858742094E-10]   ut vi inte ers  [p=3.735082555631232E-9]
rent ut vi se i [p=1.2379361686304611E-9]    ut vi inte res  [p=3.735082555631232E-9]
res inte ut vi  [p=6.826703269932555E-7]     ut vi inte ser  [p=5.976132089009971E-8]
res nu vi ett i [p=2.004998394252703E-10]    ut vi ni reste  [p=1.2190178083270168E-9]
res nu vi i ett [p=1.8907373450143707E-8]    ut vi reste in  [p=3.911327078251717E-9]
reste nu i vit  [p=3.6932941777986E-8]       ut vi ser inte  [p=7.03244119671731E-8]
reste ut i vin  [p=5.234842067974278E-8]     ve er i sitt nu [p=3.5051123568001315E-9]
reste ut vi in  [p=1.173398123475515E-8]     ve er nu i sitt [p=2.8871976198085495E-8]
```

```
reste ut vi ni  [p=5.866990617377575E-9]     vers i ett nu i [p=5.585713767774757E-9]
riset en ut vi  [p=3.956204617078708E-8]     vers i inte ut  [p=5.218269547053441E-9]
rut inte se vi  [p=1.0177793832489041E-6]    vers i nu i ett [p=1.1401286909778828E-9]
rut inte vi se  [p=2.1318476247657276E-7]    vet nu i riset  [p=1.3154700955091158E-8]
se i tur ni vet [p=1.2782933719464848E-10]   vi in reste ut  [p=3.964994124209125E-9]
se ut er tvi ni [p=7.099247719163898E-9]     vi ni reste ut  [p=8.149844519167894E-9]
se ut i vinter  [p=9.020035563278755E-7]     vi nu ser ett i [p=1.0755405029256881E-10]
sen vi i ett ur [p=3.742883064527547E-10]    vi nu ser i ett [p=3.688172702030936E-9]
sen vi i tu tre [p=3.232321138562728E-10]    vi se ut i rent [p=4.874189357955171E-10]
sen vi ut i ert [p=2.737452226214421E-10]    vi ser inte ut  [p=1.182483675258817E-7]
sen vi ut i tre [p=7.527993622089657E-10]    vi ser nu i ett [p=2.7195616893763467E-9]
ser inte ut vi  [p=8.785210787624866E-8]     vi ser ut inte  [p=7.58763392277394E-8]
ser nu vi i ett [p=5.122462086154077E-10]    vi sitter nu e  [p=3.744814625403478E-9]
ser ut en i vit [p=2.4840613417942104E-10]   vi tre i tusen  [p=2.135511044353802E-9]
ser ut i en vit [p=3.8286992714908027E-7]    vi ut inte ser  [p=7.377940850629595E-10]

Anagrams for "anagram generator":
german or a great an    [p=1.9334758773318616E-7]
german or are a gnat    [p=1.978049467675584E-9]
german or are a tang    [p=6.593498225585281E-10]
german or art an age    [p=6.422454091511403E-9]
manner o a great rag    [p=3.353617069110306E-10]
great a man or anger    [p=1.8626279303618577E-9]
marre no great an ag    [p=2.019265167212665E-9]
```

4.2 Result from using trigram probabilities

```
Anagrams for "universitet":
ser ut i en vit [p=1.186829867641766E-6]

Anagrams for "anagram generator":
*** no anagrams found ***
```

5 Conclusions

The success of any statistical method hinges on having a large enough number of samples. In the experiments done with the implementation presented in this paper, the size of the corpus was limited to about one million words due to memory issues.

Using bigram probabilities produces a reasonably good selection of anagrams. However, since it only requires each pair of words to be probable, it is often too dull an instrument and too inclusive.

Trigram probabilities take more context into account and, hence, should produce better results. However, this also increases the requirements on the corpus. Unfortunately, a corpus size of one million words proved to be too small for getting useful results using trigram probabilities since it is much more exclusive than using bigram probabilities.

A big advantage of statistical methods over formal based ones are that statistical methods are much easier to adapt to new languages or to changes in a language. In this work, both an English and a Swedish corpus were used and both of them worked to an equal degree of satisfaction without any modification to the program itself.

Statistical methods have proven to be very successful in natural language recognition, but it seems that relying solely on statistics and not taking formal semantic knowledge into consideration may be too limited a method for generating sentences. In our experiments, performance was further hampered by using a rather small corpus. This proved to be too inclusive in some respects and too exclusive in others.

Using the semantics captured in sentence probabilities for anagram generation is an improvement over merely using dictionary lookup, and drastically reduce the number of generated anagrams. However, it still produces a large number of results that has to be manually filtered. It also rejects valid but improbable anagrams like, for instance, compound nouns.

References

[1] The anagrammy awards web page. http://www.anagrammy.com/.

[2] Arrak anagram server. http://ag.arrak.fi/index_en.html.

[3] Cobuild home page. http://www.cobuild.collins.co.uk/.

[4] Corpus linguistics web page. http://www.ruf.rice.edu/ barlow/corpus.html.

[5] Internet anagram server. http://www.wordsmith.org/anagram/.

[6] Project gutenberg web site. http://promo.net/pg/.

[7] Word games source code archive. anagram source code. http://www.gtoal.com/wordgames/anagrams.html.

[8] Survey of the state of the art in human language technology. http://cslu.cse.ogi.edu/HLTsurvey/HLTsurvey.html, 1996.

[9] Christopher D. Manning and Hinrich Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 2000.

[10] Pierre Nugues. *An Introduction to Language Processing with Prolog*. to be published.

Lexical Functional Grammar: Analysis and Implementation

Magdalene Grantson

Department of Computer Science

Lund University, Box 118, 221 00 Lund, Sweden

May 16, 2002

Abstract

Lexical Functional Grammar LFG is a theory of grammar e.i. in general a theory of syntax, morphology, and semantics. It postulates two levels of syntactic representation of a sentence, a constituent structure (c-structure) and functional structure (f-structure). LFG formalism provides a simple set of devices for describing the common properties of all human Languages and the particular properties of Individual Languages. In this paper, I will describe the basic concepts underlining Lexical Functional Grammar and will implement:

- a parser for the c-structure.
- a parser for the c-structure with functional annotations

All implementations will be done in Prolog (with the use of DCF rules and list)

1 Introduction

The formalism of Lexical Functional Grammar was first introduced in 1982 by Kaplan and Bresnan after many years of research in this area. It has since been applied widely to analyse linguistics phenomena. This theory of grammar assigns two levels of syntactic representation to a sentence, the constituent structure (c-structure) and functional structure (f-structure). The c-structure have the form of context-free phrase structure trees. It serves as the basis for phonological interpretation while the f-structure are sets of pairs of attributes and values. Attributes may be features, such as tense and gender, or functions, such as subject and object. The f-structure actually encodes linguistic information about the various functional reactions between parts of a sentence. Many phenomena are thought to be more naturally analysed in terms of grammatical functions as represented in the lexicon or in f-structure, rather than on the level of phrase structure. For example, the alternation in the syntactic position in which the logical object (theme argument) appears in corresponding active passive sentences has been viewed by many linguists as fundamentally syntactic in nature and treated as transformations is handled by LFG as Lexicon. Grammatical functions are not derived from phrase structure configurations, but are represented at the parallel level of functional structure. C-structure varies across languages, however f-structure representation, which contains all necessary information for semantic interpretation of an utterance, is said to be universal.

2 Structural Representation in LFG

There are different kinds of information dependencies constituting parts of a sentence, thus giving rise to different formal structures. LFG is basically made up of *lexical structure, functional structure and constituent structure*. There are also notational conventions of LFG that a person is likely to encounter in a grammar written in LFG. The rules of LFG contain expressions known as FUNCTIONAL SCHEMATA which are symbols that appear at the right of the $--> $ arrow. The Figure 1 below shows a format for writing rules in LFG. Examples of such a format with further explanations of the arrows will be given in the next subsections.

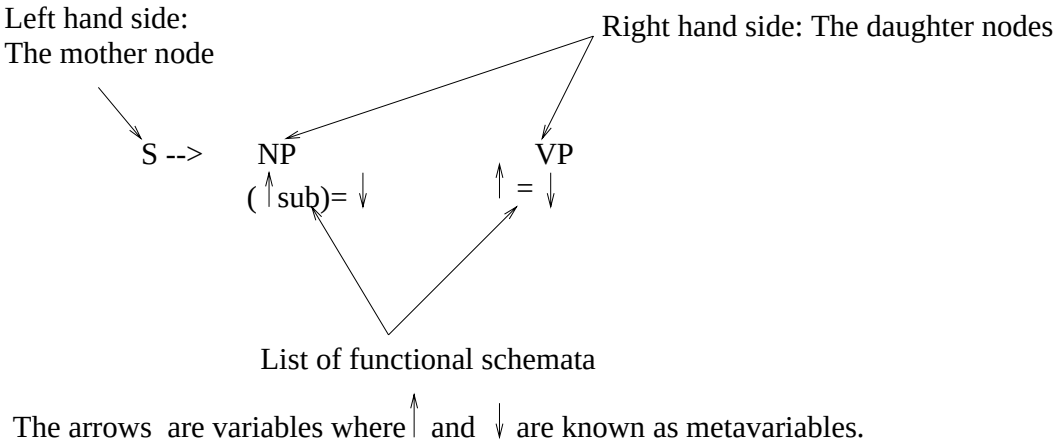

Figure 1:

2.1 Lexical structure

Lexical structures comprise information about the meaning of the lexical item, its argument structure and grammatical functions such as subject, object, etc. associated to their appropriate argument. The verb *talk* for instance, has a predicate argument structure which is made up of an agentive argument associated with the subject and a theme argument associated with the object function.

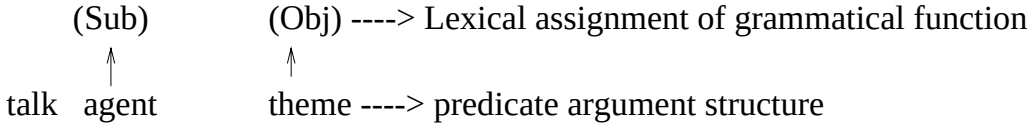

Figure 2:

Lexical items have functional schemata in LFG. Grammatical functions are associated with lexical items and syntactic positions by mean of annotated phrase structure rule.

They mediate between lexical and constituent structure representations. Grammatical functions are considered as an interface between lexicon and syntax. Below are examples of schematised formats of LFG LEXICAL ITEMS. The name **Magdalene**, for example, comes with the grammatical information such as gender.

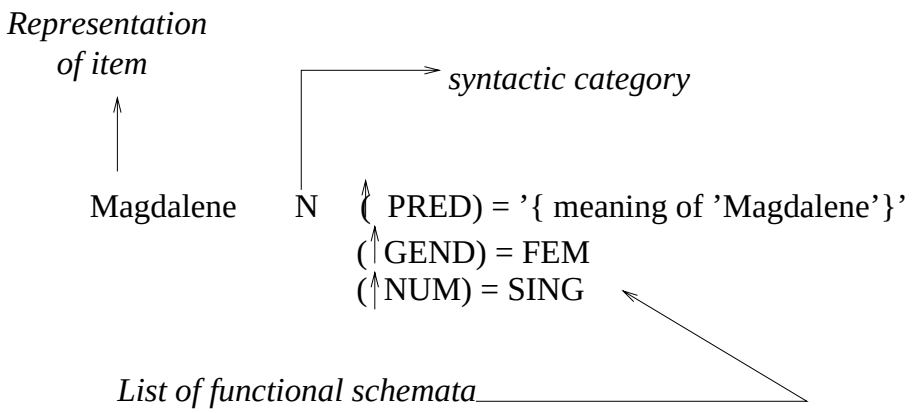

Figure 3:

A lexical rule takes a lexical item as input and returns a new lexical item. It is defines over a whole class of lexical items.

2.2 Constituent structure (c-structure)

The c-structure encodes hierarchical grouping and linear order. That is; it indicates the hierarchical structure of phrasal constituents in the string in a way familiar to most linguists. Functional annotations is functional schemata transfered into tree and is interpreted as deriving information about the functional structure.

In creating c-structures, all we need is context-free phrase-structure rules, phrase structure trees and the insertion of functional schemata. Language specific annotation of phrase-structure rules helps to identify the grammatical functions that occur in specific syntactic positions. Below is a set of phrase structure rules:

```
S -->  NP   VP
NP --> (Determiner) N (PP)
PP --> preposition NP
VP --> V ( NP) (NP) PP
```

If we consider the sentence **Magda likes Ann**; we first of all consider the syntactic rules of this sentence, then we construct a tree with annotations prescribing the rules. Figure 4 shows a relation between rules and annotations in the tree.

The arrow ↑ Refers to the f-structure (see next subsection) of the mother node. It is instantiated *(Instantiation transforms the schemata into functional equation)* by the node immediately dominating the constituent under which the arrow is placed. The arrow ↓ refers to the f-structure of the current node. Thus from the rule S $--$ >NP VP, the equation states that NP is the subject (Sub) of S that dominates it. The $\uparrow = \downarrow$ equation

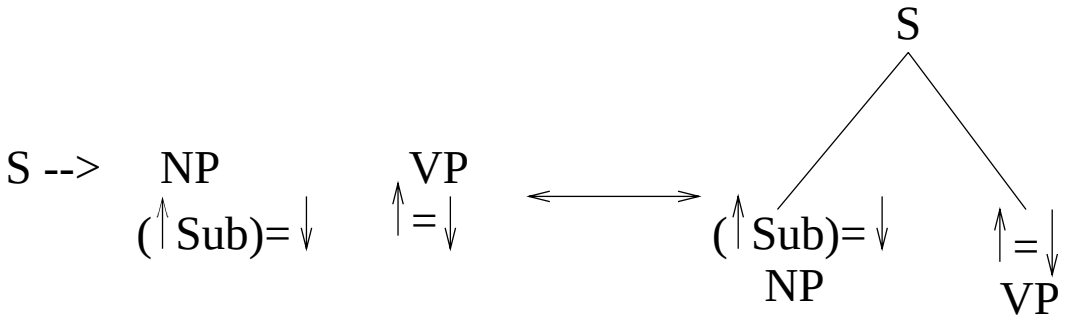

Figure 4: Relation between rules and annotations in the tree

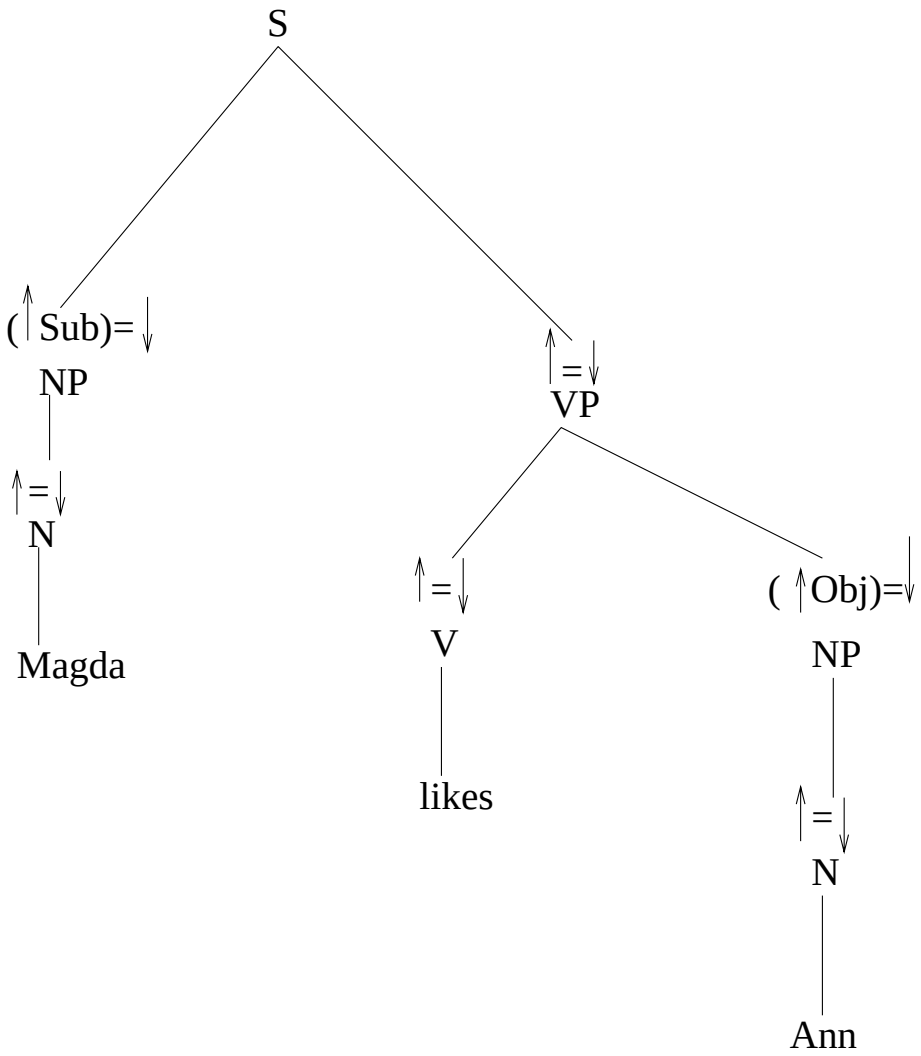

Figure 5:

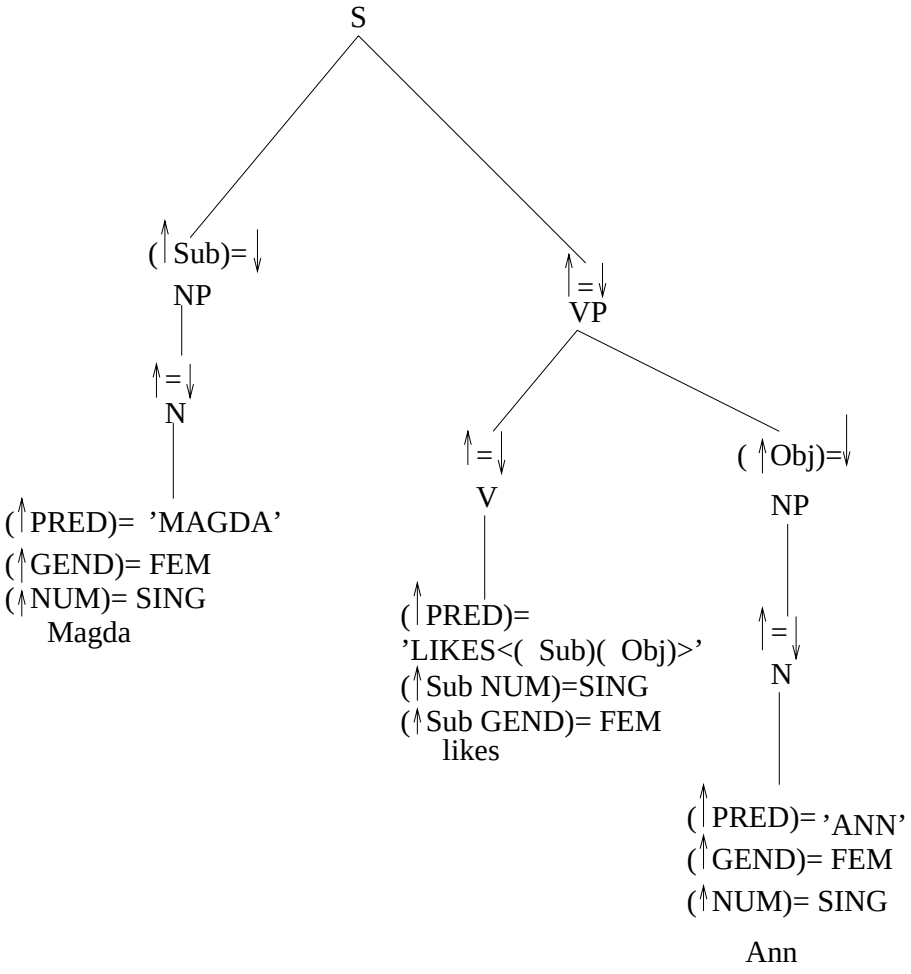

Figure 6:

beneath VP indicates that the features of that node are shared with higher nodes, making all functional information carried by this node also direct information about the mother's f-structure. (↑ SUB)= ↓) means all functional information carried by this node goes into the subject part of the mother's function. Thus from the sentence **Magda likes Ann** the next stage of the tree will be like indicated in Figure 5.

The c-structure is complete after introducing the annotations specified by the lexical entries for **Magda likes Ann**. This is achieved by consulting the lexical entry for each lexical item in the tree for their functional schemata. Figure 6 is the annotated c-structure for the sentence **Magda likes Ann**

2.3 Functional Structure (f-structure)

Functional structures are unification-based grammars and encode information about the various functional reactions between parts of sentences. They are themselves functions from attributes to value, where the value for an attribute can be:

- Atomic symbols eg. [NUM SING]

- Semantic form eg. [PRED 'TRUST¡(↑ S (↑ OBJ)¿]
- f-structure, eg.

$$\begin{bmatrix} Sub \begin{bmatrix} PRED & 'MAGDA' \\ NUM & SING \\ GEND & FEM \end{bmatrix} \end{bmatrix}$$

Graphical f-structures are represented in this paper as material enclosed in large square brackets as in Section 3. There is a basic assumption in LFG, which states that there is some f-structure associated with each node in the constituent structure tree. Hence lexical information is combined with structural information available from the c-structure tree to get the f-structure.

3 Well-formedness conditions on F-structure

The f-structure is valid according to well-formedness condition. Well-formedness condition filter out over generation in c-structures

3.1 Functional Uniqueness

Functional uniqueness is also refered to as consistency. It ensures that each f-structure is a function whereby each attribute has a unique value.

Example of consistent structure in the f-structure

$$\begin{bmatrix} NUM & SING \\ NUM & PL \end{bmatrix} \begin{bmatrix} NUM \end{bmatrix}$$

Example of an inconsistent structure in the f-structure

$$\begin{bmatrix} GEND & FEM \\ GEND & MAL \end{bmatrix}$$

3.2 Completeness

Completeness ensures that sub-categorization requirements are met. An f-structure is not well formed if it lacks values for grammatical functions that are sub-categorized by the predicate. The example below lacks a value for the subject (Sub) and is therefore termed as incomplete.

eats

3.3 Coherence

Coherence ensures that every argument is the argument of some predicate.

4 Implementation

The implementation is done in Prolog, with the use of DCG rules and Prolog terms. It is based on ideas from the lecture notes from Pierre Nugues' Linguistics course and [5]. The implementation of a parser for c-structure is given at Appendix A. A list of sentences provided in Section 5 are used so as to produce an appropriate c-structure. A sentence like **I have a bag** can be queried as :

```
?-parser(Cstructure,[i,have,a,bag])
```

to get the appropriate c-structure:

```
L = s(np([pron(i)]), vp([v(have), np([det(a), n(bag)])])) .
```

In Appendix B, an implemented demonstration of an LFG grammar (c-structure with functional annotation) is given. The check for completeness and coherence in the f-structure is not considered in the implementation. A sentence like **Magdalene washed the orange** can be queried as :

```
?- parser(Cstucture,Fstructure,[magdalene,washed,the,orange]).
```

```
to get the c-structure and f-structure as:

Cstucture = s(np(pn(magdalene)), vp(v(washed), np(det(the), n(orange))))
Fstructure = fs(pred:wash, spec:_G600, person:third, numb:sg, tense:past,
subj:fs(pred:magdalene, spec:_G645, person:third, numb:sg, tense:_G654,
subj:_G657, obj:_G660, obj2:_G663, ajunct:_G666, pcase:_G669),
obj:fs(pred:orange, spec:the, person:third, numb:sg, tense:_G787, subj:_G790,
obj:_G793, obj2:_G796, ajunct:_G799, pcase:_G802), obj2:_G618, ajunct:_G621,
pcase:_G624)

Yes
```

5 Sentences

5.1 Sample sentences for program in appendix A

```
I have a bag.
I give the bag to Peter.
Peter sells George the bag.
I walk into an Irish pub.
```

```
I order a drink.
I like soft drinks.
I dance everyday.
I had a dance yesterday.
Jane like dates.
Jane dated George.
```

5.2 Sample sentences for program in appendix B

```
Magdalene washed a dress.
James washed an orange.
Magdalene washed the plates.
```

6 Conclusion

Lexical Functional Grammar (LFG) was first documented in 1982 by Joan Bresnan.[1] It has three levels of representation,each having different formal characterization. These are Lexical structure, Constituent structure and Functional structure. The single level of syntactic representation c-structure, exits simultaneously with an f-structure representation that integrates the information from c-structure and from lexicon.

References

[1] Bresnan J (ed) 1982. The mental representation of grammatical relations. MIT Pree,Cambridge, Massachusetts.

[2] Kaplan, Ronald M and Bresnan J. 1982. Lexical-Functional grammar: a formal system for grammatical representation.emph Bresnan 1982.

[3] Hopcroft, John E. J.D. Ullman. 1979. Introduction to automata theory, languages and computation. Reading mass. Addison Wesley

[4] Kaplan R. , Ronald M, Maxwell J.T. 1993. LFG Grammar Writer's Workbench. Xeror Palo Alto Research Center.

[5] Clocksin W, Mellish C., Programming in Prolog.1994, Springer

[6] Pierre Nugues, 2001 lecture notes, An outline of theories, Implementations, and applications with special consideration of English, French, and German.

Appendix A

```
/*-------------------------------------------------------------
                          PARSER

This is a predicate that passes a list of words to produces an
appropriate c-structure

-------------------------------------------------------------*/
parser(L1,L):-
        s(L1,L,[]).

/*-------------------------------------------------------------

                           GRAMMAR
-------------------------------------------------------------*/

% Sentence
s(s(NP,VP)) --> np(NP), vp(VP).

% Prepositional Phrase
pp(pp(P,NP)) --> p(P), np(NP).




% noun phrase

np(np(NP)) --> np0(NP).

np0([Pron|NP5]) --> pronoun(Pron), np5(NP5).
np0([PropN|NP5]) --> proper_noun(PropN), np5(NP5).
np0([Det|NP1]) --> det(Det), np1(NP1).
np0([Adj|NP3]) --> adj(Adj), np3(NP3).
np0([PossP|NP1]) --> possessive_pronoun(PossP), np1(NP1).

np1([N|NP5]) --> noun(N), np5(NP5).
np1([Adj|NP2]) --> adj(Adj), np2(NP2).

np2([N|NP5]) --> noun(N), np5(NP5).

np3([PN|NP5]) --> proper_noun(PN), np5(NP5).

np5([],L,L).
```

```
%verb phrase


vp(vp(VP)) --> vp0(VP).

vp0([V|VP1]) --> verb(V), vp1(VP1).

vp1([NP|VP3]) --> np(NP), vp3(VP3).
vp1([PP|VP4]) --> pp(PP), vp4(VP4).
vp1([NP|VP2]) --> np(NP), vp2(VP2).

vp2([NP|VP3]) --> np(NP), vp3(VP3).

vp3([],L,L).
vp3([Adv|VP5]) --> adv(Adv), vp5(VP5).
vp3([PP|VP4]) --> pp(PP), vp4(VP4).

vp4([],L,L).
vp4([Adv|VP5]) --> adv(Adv), vp5(VP5).

vp5([],L,L).



adj(adj(Adj)) -->
        [Adj],
        {adj(Adj)}.

adv(adv(Adv)) -->
        [Adv],
        {adv(Adv)}.

det(det(Det)) -->
        [Det],
        {det(Det)}.

noun(n(N)) -->
        [N],
        {n(N)}.

p(prep(Prep)) -->
        [Prep],
        {prep(Prep)}.

possessive_pronoun(poss_pron(her)) --> [her].
```

```
pronoun(pron(Pron)) -->
        [Pron],
        {pron(Pron)}.

proper_noun(propn(PropN)) -->
        [PropN],
        {prop_n(PropN)}.

verb(v(V)) -->
        [V],
        {v(V)}.



%  Lexicon

% Adjectives
adj(irish).
adj(soft).

% Adverbs
adv(everyday).
adv(yesterday).

% Determiners
det(a).
det(the).
det(an).

% Nouns
n(bag).
n(pub).
n(dance).
n(drink).
n(walk).
n(car).
n(competition).

% Prepositions
prep(for).
prep(in).
prep(to).
prep(into).

% Pronouns
pron(she).
pron(i).
```

```
prop_n(dates).
prop_n(drinks).

prop_n(peter).
prop_n(george).
prop_n(jane).

% Verbs
v(order).
v(dated).
v(have).
v(had).
v(give).
v(sells).
v(like).
v(likes).
v(dance).
v(walk).
```

Appendix B

```
/*-------------------------------------------------------------
                          PARSER

This is a predicate that passes a list of words to  produces an
appropriate c-structure and f-structure

-------------------------------------------------------------*/


parser(Cstruc,Fstruc,Ins):-
        s(Cstruc,Fstruc,Ins,[]).

% S --> NP, VP.
%
s(s(NP,VP),fs(pred : Pred,
                spec : Spec, person : Person, numb : Numb, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) -->
        np(NP,Subj),
```

```
        vp(VP,fs(pred : Pred,
                spec : Spec, person : Person, numb : Numb, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)).

% Noun Phrase

% NP --> Determiner, N.

np(np(Det,N),FS) -->
        det(Det,FS),
        n(N,FS).

% NP --> Proper noun

np(np(PN),FS) -->
        pn(PN,FS).

% Verb Phrase

% VP --> V.

vp(vp(V),fs(pred : Pred,
                spec : Spec, person : Person, numb : Numb, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) -->
        v(V,fs(pred : Pred,
                spec : Spec, person : Person, numb : Numb, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)).

% VP --> V, NP.

vp(vp(V,NP),fs(pred : Pred,
                spec : Spec, person : Person, numb : Numb, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) -->
        v(V,fs(pred : Pred,
                spec : Spec, person : Person, numb : Numb, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)),
        np(NP,Obj).

% V --> V, NP, NP.

vp(vp(V,NP1,NP2),fs(pred : Pred,
                spec : Spec, person : Person, numb : Numb, tense : Tense,
```

```
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) -->
        v(V,fs(pred : Pred,
                spec : Spec, person : Person, numb : Numb, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)),
        np(NP1,Obj),
        np(NP2,Obj2).


% Lexicons

det(det(the), fs(pred : Pred,
                spec: the, person : Person, numb : Numb, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) --> [the].

det(det(a), fs(pred : Pred,
                spec: a, person : Person, numb : sg, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) --> [a].

det(det(an), fs(pred : Pred,
                spec: a, person : Person, numb : sg, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) --> [an].

pn(pn(magdalene), fs(pred : magdalene,
                spec: Spec, person : third, numb : sg, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) --> [magdalene].

pn(pn(james), fs(pred : james,
                spec: Spec, person : third, numb : sg, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) --> [james].

n(n(dress), fs(pred : dress,
                spec: Spec, person : third, numb : sg, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) --> [dress].

n(n(orange), fs(pred : orange,
                spec: Spec, person : third, numb : sg, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) --> [orange].
```

```
n(n(plates), fs(pred : plate,
                spec: Spec, person : third, numb : pl, tense : Tense,
                subj : Subj, obj : Obj, obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) --> [plates].

v(v(washed), fs(pred : wash,
                spec : Spec, person : Person, numb : Numb, tense : past,
                subj: fs(pred : SPred,
                         spec : SSpec, person : Person, numb : Numb,
                         tense : STense,
                         subj : SSubj,
                         obj : SOBJ, obj2 : SObj2,
                         ajunct : SAjunct, pcase : SPCase),
                obj : Obj,
                obj2 : Obj2,
                ajunct : Ajunct, pcase : PCase)) --> [washed].
```

Selecting an Appropriate Language Base for Automated Requirements Analysis

In companies that constantly develop new software releases for large markets, new requirements written in natural language continuously arrive that may affect the current development situation. Before making any decision about the requirements, they must be somewhat analysed and understood and related to the current set of implemented and waiting requirements. The task is time-consuming due to the high inflow of requirements and any support that may help the requirements analysts and reduce labour would provide faster and improved decision-support. This paper investigates the terms used in software requirements written in natural language. The investigated requirements come from a real industrial setting and have been used for 5 years when developing subsequent releases of a software application. Through term extraction, sets of requirements from different years, are compared to each other, to the BNC Sampler, and to the documentation of the application. The results show that (1) the language used in requirements may be considered to be unique compared to general spoken and written language, (2) the requirements and the documentation share basically the same domain-specific words, whereas the most used phrases differ, and (3) the language in the requirements is fairly consistent over the years. The results indicate that previous requirements is the better source, compared both to general language and documentation of the software, when building tools based on natural language processing techniques to analyse requirements.

1. Introduction

In software development, requirements are an important means for meeting the needs of the customer. The requirements may be identified and specified either before the specific software is designed, forming a point of departure for the software engineer, or after the software has been tested or used for some time, thus forming a basis for correcting and improving the software and to meet the needs and complaints of the customer.

In traditional software development, also known as bespoke software development, requirements are usually negotiated and agreed upon with a known end user or customer, before development is pursued or finished (Sawyer, 2000). Thus, the devloping company and the customer together identify and specify the requirements. Further, the customer may provide feedback during development to influence which particular requirements should be implemented.

In market-driven software development, there is very limited negotiation with end users and customers (Potts, 1995; Carlshamre, 2002). Rather, requirements are invented and specified in-house. Thus, the developing company relies on employees providing appropriate requirements. The employees are acting as *stakeholders* for different parts of the organisation. The stakeholders include, for example, marketing department, support, and development, that provide requirements of different kinds, such as bug reports, suggestions for new functionality, suggestions for improving already existing functionality, etc.

In order not to miss any good ideas, anyone within the organisation may submit a requirement and requirements are collected continuously during development. The requirements are most often written in natural language and stored in some kind of repository or database. Reading and classifying, i.e. analysing, the requirements are time-consuming, tiresome tasks and the amount of requirements thus easily becomes overflowing and requirements analysis and manangement may therefore entail bottlenecks in the development process.

Different attempts at performing the requirements analysis more or less automatically have been undertaken (Natt och Dag, Regnell, Carlshamre, Andersson, & Karlsson, 2002, pp. 21–22). These automated techniques could support the requirements analyst by suggesting duplicates and groupings based on the requirements' linguistic content. To perform a deeper linguistic analysis of software requirements, an adequate lexicon is crucial. The lexicon must be domain-specific and also contain everyday vocabulary. Such a lexicons is usually constructed manually since there is, to our knowledge, no other way available today. This makes the techniques rather expensive and less appealing to the software development company. In particular, the market-driven organisation does not interpret requirements the same way as in customer-specific development. In the latter, developers usually have very good domain expertise, whereas the former more often rely on consultants and

Johan Natt och Dag, Dept. of Communication Systems, Lund University, johan.nattochdag@telecom.lth.se
Kerstin Lindmark, Dept. of Linguistics, Lund University, kerstin.lindmark@ling.lu.se

brainstorming sessions (Lubars, Potts, & Richter, 1993). Therefore, it is even more troublesome to manually construct an appropriate domain-specific lexicon.

This paper explores the possibility of constructing a domain-specific lexicon more or less automatically. It is hypothesised that requirements or documentation already available in the organisation may be used as a language base when extracting terms for constructing an appropriate lexicon. The properties of three possible sources for term extraction are investigated. Firstly, the BNC Sampler, which is a two percent sample of the full British National Corpus representing both written and spoken language (one million words per category). Secondly, the documentation of a software application developed by a market-driven software development company. Thirdly, the requirements from the development of several releases of the above-mentioned software application.

This paper is organised as follows: In Section 2, the data sources used in the analysis are described. In Section 0, the techniques used are explained, and how the anlysis was carried out is elaborated in Section 4. In Section 5, the raw results are presented and, finally, in Section 6, the conclusions from the results are presented.

2. Data sources

Three data sources have been analysed and compared, with respect to the vocabulary used:

- *The BNC Sampler*
  The British National Corpus (Burnard, 2000) is a 100 million word corpus of modern British English, both spoken and written. It was created in 1994 by a consortium of leading dictionary publishers and research institutions. From the full BNC, a two percent sample has been created and called the *BNC Sampler*. The BNC Sampler consists of one million words spoken, and one million words written. More information is available at http://info.ox.ac.uk/bnc.

- *Software documentation*
  The documentation (Telelogic Tau 4.2 Documentation, 2001) is written in regular English and should also comprise domain specific words. The documentation used belongs to version 4.2 of the software application and is thus rather mature. In total, it comprises 1,069,478 words (called tokens), of which 14,085 are unique (called types). In Table 1 the number of words of different lengths are shown.

Table 1. Number of words of different lengths in the software documentation.

1-letter words	64,565
2-letter words	171,960
3-letter words	217,433
4-letter words	166,958
5-letter words	94,492
6-letter words	95,828
7-letter words	81,950
8-letter words	56,400
9-letter words	54,228
10-letter words	28,054
11-letter words	18,960
12-letter words	6,935
13-letter words	5,008
14(+)-letter words	2,825
Total	**1,065,596**

The totals differ when counting all words and summing up the counts of different word lengths. The software used (WordSmith) does not explain this, nor has a plausible explanation been found. Fortunately the difference is relatively small (4,118 words, which comprise a 0,36 % difference) compared to the full set.

- *Software Requirements*
  From the software vendor we received a database comprising 1,932 requirements written in natural language. The majority of the requirements are written in English, irrespective of the authors' mother tongue. Thus, the quality varies and in some requirements there are also Swedish phrases. Example requirements, with all the attributes the company uses, can be found in Table 3a and Table 3b.

  Due to the continuous elicitation, the requirements concern different stages of development, such as elicitation, selection, and construction. Thus, the requirements are analysed to various degrees. The requirements have been collected during five years of development, starting in 1996. In Table 2 the number of requirements

Johan Natt och Dag, Dept. of Communication Systems, Lund University, johan.nattochdag@telecom.lth.se
Kerstin Lindmark, Dept. of Linguistics, Lund University, kerstin.lindmark@ling.lu.se

Table 3a. Example requirement submitted 1996.

RqId	RQ96-270
Date	
Summary	Storing multiple diagrams on one file
Why	It must be possible to store many diagrams on one file. SDT forces to have 1 diagram per file. It's like forcing a C programmer to have not more than one function per file... The problem becomes nasty when you work in larger projects, since adding a procedure changes the system file (.sdt) and you end up in a mess having to "Compare systems".
Description	Allow the user to specify if a diagram should be appended to a file, rather than forcing him to store each diagram on a file of its own.
Dependency	4
Effort	4
Comment	This requirement has also been raised within the multiuser prestudy work,but no deeper penetration has been made. To see all implications of it we should have at least a one-day gathering with people from the Organizer, Editor and InfoServer area, maybe ITEX? Här behövs en mindre utredning, en "konferensdag" med förberedelser och uppföljning. Deltagare behövs från editor- och organizergrupperna, backend behövs ej så länge vi har kvar PR-gränssnittet till dessa.
Reference	
Customer	All
Tool	Don't Know
Level	Slogan
Area	Editors
Submitter	x
Priority	3
Keywords	storage, diagrams, files, multi-user
Status	Classified

Table 3b. Example requirement submitted 997. This is actually a duplicate of the requirement in Table 3.

RqId	RQ97-059
Date	Wed Apr 2 11:40:20 1997
Summary	A file should support storing multiple diagrams
Why	ObjectGeode has it. It's a powerful feature. It simplifies the dayly work with SDT. Easier configuration management. Forcing one file for each procedure is silly.
Description	The SDT “Data model" should support storing multiple diagram on one file.
Dependency	4
Effort	1-2
Comment	Prestudy needed
Reference	http://info/develop/anti_og_package.htm
Customer	All
Tool	SDT SDL Editor
Level	Slogan
Area	Ergonomy
Submitter	x
Priority	3: next release (3.3)
Keywords	diagrams files multiple
Status	Classified

from each year is shown together with the number of words they comprise.

Table 4. Number of requirements from the different years and the number of words they comprise.

Year	Requirements	Words
1996	459	34,588
1997	714	60,944
1998	440	40,099
1999	239	20,029
2000	80	7,911
Total	**1,932**	**163,571**

3. Term extraction

The process of establishing a terminology for a given domain involves not only a meticulous semantic analysis of terms and definition writing, but also extraction of terms from a relevant material (Suonuuti, 2001). Traditionally, this is a time-consuming business, involving a lot of manual work.

Whereas a terminologist normally spends a lot of time reading the material and trying to figure out which words are typical for the domain, we decided to adopt a more mechanical appoach. In this paper, corpus lingustic analysis methods are used to find the words and phrases which could be considered as domain specific terms, and also general language expressions that appear to be used in a specific way, or simply to be over-represented.

Johan Natt och Dag, Dept. of Communication Systems, Lund University, johan.nattochdag@telecom.lth.se
Kerstin Lindmark, Dept. of Linguistics, Lund University, kerstin.lindmark@ling.lu.se

Corpus lingustic analysis methods involve the following:

- The texts to be analysed are collected into one or several corpora.

- Appropriate software tools, such as Corpus Work Bench, QWICK, SARA and WordSmith, are used to produce different statistics of the corpora:
  - *Word frequency lists*
  - *Keywords* (comparing frequency lists)
  - *Concordances* (contexts in which a given word occurs)
  - *n-grams* (strings of *n* words)
  - Lengths and ratios of different units within the corpora (such as sentences, paragraphs, etc.)

In this paper, WordSmith was used to perform all the analyses, as it provides fast and accurate tools for processing large corpora. The statistics presented in this paper include word frequency lists, keywords, and n-grams. Keywords may require some explanation as these have been used differently to measure similarity between corpora. In WordSmith a keyword is defined as follows:

"a) it occurs in the text at least as many time as the user has specified as a *minimum frequency*.

b) its frequency in the text when compared with its frequency in a reference corpus is such that the statistical probability as computed by an appropriate procedure is smaller than or equal to a p value specified by the user."

(WordSmith Tools Help)

The "key-ness" of a word in the corpus is determined by calculating the frequencies and the number of words in both corpora. These are then cross-tabulated and two statistical tests of significance are computed: a χ^2 test (Siegel & Castellan, 1988) and Ted Dunning's Log Likelihood test (Dunning, 1993). The latter gives a better estimate of keyness, especially when comparing long texts or whole genres against reference corpora. Other measures of similarity, using word frequency lists, are discussed by Kilgarriff (1997).

4. Analysis

From the requirements corpus a simple frequency list was first produced. This list was compared with the frequency list from the BNC Sampler corpus, using the Keywords feature in the WordSmith Tools. The resulting list of 500 words, whose frequency in the requirements word list is significantly higher than in the BNC Sampler word list, was then scrutinised manually.

The same procedure was repeated with, in turn, the documentation vs. the BNC Sampler list, the requirements vs. the documentation, subsets of requirements (the requirements from a certain year were extracted as a subset) vs. the whole set of requirements, the subsets vs. each other, each subset vs. a set consisting of the other subsets, and, for each of these comparisons, a comparison in the opposite direction, i.e. the documentation vs. the requirements and so forth.

It is, of course, to be expected that quite a number of the terms used in the requirements are multiwords. To investigate this, bigrams, trigrams, and tetragrams were extracted. This was done using Perl scripts. The amount of multiwords yielded by simply extracting strings of two, three or four words obviously exceeded manageable numbers by far. To clean up, different scripts were constructed to sort out unique occurrences, delete lines starting by numbers, punctuation marks, parentheses and the like. Finally, lines starting with prepositions and conjunctions were deleted.

The number of multiwords retained is still very large, and the next step would be part-of-speech-tagging the corpus so that the set of term candidates can be limited to strings containing at least one noun.

At this stage we decided to try out the *n*-gram finding feature in the WordSmith Tools instead, called "clusters". As a default, this function finds *n*-grams with a frequency of at least two, and since this turned out to give neat, manageable lists of apparently relevant words and phrases, we decided to use these. The lists of bi-, tri- and tetragrams were compared to each other with the keyword feature in a way corresponding to what was done with the single words.

One of the crucial questions for this investigation is whether the terminology of the requirements is consistent throughout the different sets, making it possible to "predict the linguistic content" of the whole set of requirements based on an analysis of a subset, and, even more important, of new requirements based on an analysis of the old ones. To investigate this, the requirement were divded into subsets, according to year of origin. All

Johan Natt och Dag, Dept. of Communication Systems, Lund University, johan.nattochdag@telecom.lth.se
Kerstin Lindmark, Dept. of Linguistics, Lund University, kerstin.lindmark@ling.lu.se

possible different combinations of subsets were analysed. In brief, comparisons of one subset to the rest of the requirements yield a small number of keywords; in the case of bi-, tri-, and tetragrams even none, in quite a few cases. The small number can be even further reduced, if only words that could be term candidates are retained. As for the comparison of one subset with a set consisting of all the other sets, i.e. the set under investigation not included itself, this yields a somewhat larger number of keywords; approximately thirty per set.

A complete overview of all the comparisons that were found relevant for the questions and conclusions in this paper is found in Table 5. The relevance was decided upon afterwards. The other comparisons made were used to obtain an understanding of the nature of the corpora and how they are related to each other. The complete resulting data for the analysis can be found in an Excel workbook at http://www.telecom.lth.se/Personal/johannod/education/cling/AnalysisResults.xls.

Table 5. Overview of comparisons made between the different corpora and subsets of the corpora.

Id	Corpus under investigation	Reference corpus
req→bnc	All requirements	BNC Sampler
req→doc	All requirements	Documentation
doc→bnc	Documentation	BNC Sampler
2req→2bnc	Bigrams All requirements	Bigrams BNC Sampler
4req→4doc	Tetragrams All requirements	Tetragrams Documentation
97→96	Requirements 1997	Requirements 1996
98→96-97	Requirements 1998	Requirements 1996-1997
99→96-98	Requirements 1999	Requirements 1996-1998
00→96-99	Requirements 2000	Requirements 1996-1999
96→Rest	Requirements 1996	All requirements but 1996
97→Rest	Requirements 1997	All requirements but 1997
98→Rest	Requirements 1998	All requirements but 1998
99→Rest	Requirements 1999	All requirements but 1999
00→Rest	Requirements 2000	All requirements but 2000

Complete resulting data can be found at
http://www.telecom.lth.se/Personal/johannod/
education/cling/AnalysisResults.xls

5. Results and anlysis

Already the keywords list resulting from the comparison of the full set of requirements and the BNC Sampler shows some interesting features of the linguistic contents of these texts (Table 6, left column). Among the most significant words are not only domain-specific terms, but overrepresented words with a high frequency in general language, such as *should*, *be,* and *is* (i.e. these are not considered terms at all. The words are boldfaced in the table and those that are adjectives are also italicized). This is not very surprising, considering that the requirements are about features that are not working or features that the requirement stakeholder wishes to have. The comparison between the requirements and the documentation points even clearer in the same direction, where *I, should, would, etc.,* are overrepresented words in the requirements (Table 6, right column).

Table 6. Truncated keywords lists from single word comparison between all the requirements, on the one hand, and the BNC Sampler and the documentation, respectively, on the other hand.

N	req→bnc	req→doc
1	SDL	**I**
2	SDT	**SHOULD**
3	**SHOULD**	**WOULD**
4	SYMBOL	SDT
5	FILE	**IT**
6	MSC	**WE**
7	EDITOR	**TO**
8	ORGANIZER	TODAY
9	USER	**OUR**
10	DIAGRAM	**HAVE**
11	FILES	CUSTOMERS
12	SYMBOLS	ITEX
13	CODE	DOCUMENTATION
14	TEXT	SUPPORT
15	ITEX	**LIKE**
16	**BE**	USER
17	MENU	MINISYSTEM
18	DIAGRAMS	***NICE***
19	SIMULATOR	**THINK**
20	PAGE	**DON'T**
21	DIALOG	**ABLE**
22	TOOL	**MAKE**
23	***POSSIBLE***	**VERY**
24	TAU	***EASIER***
25	**IS**	***BETTER***

The overrepresentaion of expressions common in general language is shown even more clearly in the tetragram comparison (Table 7).

Johan Natt och Dag, Dept. of Communication Systems, Lund University, johan.nattochdag@telecom.lth.se
Kerstin Lindmark, Dept. of Linguistics, Lund University, kerstin.lindmark@ling.lu.se

Table 7. Truncated keywords lists from tetragram comparison between all the requirements and the documentation.

N	4req→4bnc
1	SHOULD BE POSSIBLE TO
2	IT SHOULD BE POSSIBLE
3	TO BE ABLE TO
4	ALLOW THE USER TO
5	IT WOULD BE NICE
6	SHOULD BE ABLE TO
7	IN THE DRAWING AREA
8	THERE SHOULD BE A
9	IS NOT POSSIBLE TO
10	IN THE SDL EDITOR
11	IT IS NOT POSSIBLE
12	MAKE IT EASIER TO
13	WOULD BE NICE TO
14	DEFECT POSTPONED IN #
15	PM DEFECT POSTPONED IN
16	WANT TO BE ABLE
17	IN THE MSC EDITOR
18	THE USER WANTS TO
19	SHOULD BE EASY TO
20	THE USER HAS TO
21	TO MAKE IT EASIER
22	IT IS POSSIBLE TO
23	I WOULD LIKE TO
24	LIKE TO BE ABLE
25	END # #
26	BE POSSIBLE TO USE
27	BE NICE TO HAVE
28	WOULD LIKE TO HAVE
29	THE PROBLEM IS THAT
30	MAKE IT POSSIBLE TO

By comparing the tetragrams from the requirements to to those in the documentation, it can be further established that the differences are not due to the particular domain, but rather to the type of text. The results from the comparison is shown in Table 8.

Table 8. Truncated keywords lists from tetragram comparison between all the requirements and the documentation.

N	4req→4doc
1	SHOULD BE POSSIBLE TO
2	IT SHOULD BE POSSIBLE
3	TO BE ABLE TO
4	IT WOULD BE NICE
5	I WOULD LIKE TO
6	SHOULD BE ABLE TO
7	ALLOW THE USER TO
8	WOULD BE NICE TO
9	THERE SHOULD BE A
10	DEFECT POSTPONED IN #
11	WOULD LIKE TO HAVE
12	PM DEFECT POSTPONED IN
13	MAKE IT EASIER TO
14	WANT TO BE ABLE
15	TO MAKE IT EASIER
16	LIKE TO BE ABLE
17	THE PROBLEM IS THAT
18	BE NICE TO HAVE
19	END # #
20	IT WOULD BE VERY
21	WOULD LIKE TO BE
22	ID WAS # #
23	PREVIOUS ID WAS #
24	IT SHOULD BE EASY
25	WOULD BE NICE IF
26	I WANT TO BE
27	SHOULD BE EASY TO
28	USER SHOULD BE ABLE
29	THE USER WANTS TO
30	BE POSSIBLE TO USE

As for the actual domain-specific terms, these appear rather from the mono- and bigram lists (and, although to a lesser extent, also from the trigram lists, not shown here). In Table 9, the results from comparing single words lists of the requirements and the documentation, respectively, with the BNC Sampler are shown. As shown in Table 10, by comparing bigrams, domain-specific compound terms may be captured, although there are many non-useful combinations.

Table 9. Truncated keywords lists from single word comparison between all the requirements and the documentation, respectively, on the one hand, and the BNC Sampler, on the other hand.

N	req→bnc	doc→bnc
1	SDL	SDL
2	SDT	TELELOGIC
3	**SHOULD**	TAU
4	SYMBOL	PAGE
5	FILE	TYPE
6	MSC	FILE
7	EDITOR	USER'S
8	ORGANIZER	TTCN
9	USER	C
10	DIAGRAM	MANUAL
11	FILES	**MARCH**
12	SYMBOLS	UM
13	CODE	CHAPTER
14	TEXT	SUITE
15	ITEX	TEST
16	**BE**	SIGNAL
17	MENU	SYMBOL
18	DIAGRAMS	**IS**
19	SIMULATOR	DIAGRAM
20	PAGE	MENU
21	DIALOG	SYSTEM
22	TOOL	PROCESS
23	***POSSIBLE***	CODE
24	TAU	NAME
25	**IS**	COMMAND

Johan Natt och Dag, Dept. of Communication Systems, Lund University, johan.nattochdag@telecom.lth.se
Kerstin Lindmark, Dept. of Linguistics, Lund University, kerstin.lindmark@ling.lu.se

Table 10. Truncated keywords lists from bigram comparison between all the requirements and the BNC Sampler.

N	2req→2bnc
1	**SHOULD BE**
2	**THE USER**
3	**THE ORGANIZER**
4	**POSSIBLE TO**
5	**THE SDL**
6	**THE TEXT**
7	**IT SHOULD**
8	**BE POSSIBLE**
9	**THE MSC**
10	**THE SIMULATOR**
11	**IN THE**
12	**TO USE**
13	**BE ABLE**
14	MACRO MINISYSTEM#
15	**AN SDL**
16	**IS NOT**
17	**THE SYMBOL**
18	SDL SYSTEM
19	**THE EDITOR**
20	**ABLE TO**
21	**THE FILE**
22	**THE SDT**
23	**IN SDT**
24	**IN SDL**
25	**POSSIBILITY TO**
26	**USER TO**
27	SDL EDITOR
28	CODE GENERATOR
29	**THE TOOL**
30	**SDT #**

Several questions were asked before starting out with the analyses, of which two may now be partly answered:

1. *Is there a specific language for requirements?*
   Yes. Firstly, the keywords list compared to the BNC Sampler comprises many domain-specific words (Table 6). Although this is not surprising, the result thus validates the expected. Secondly, the tetragram comparison (between requirements and the BNC Sampler) shows that certain phrases are used more often in requirements than in general language (Table 7). Thirdly, these phrases are not due to the domain, but, rather, to the nature of requirements (Table 8).

2. *Does the language differ or overlap between the documentation and the requirements?*
   Overlap, with respect to domain-specific terms. Differ, with respect to other lingustic features. Comparing both the keywords lists of the requirements and of the documentation to the BNC Sampler shows that the same domain-specific terms occur in both the requirements and the documentation (Table 9). The tetragram comparison between requirements and the documentation has, as already discussed above, shown that the phrases are due to the nature of requirements. Thus, requirements differ in the use of phrases (Table 8).

As for the question of the consistency of requirements throughout the different sets, some relevant results are shown in Table 11 and Table 12. Table 11 list the terms that are significantly more and less common, respectively, in the requirements from 1996 compared to those from 1997. Table 12 shows the corresponding results from comparing the requirements from 1996 to those from all the other years.

It can be seen that some terms are unique for the 1996 set (terms marked with an asterisk). However, most of them may be disregarded as domain-specific terms. It may be concluded from these lists that the words used in requirements are quite consistent over the years.

A very interesting result is that the lists are relatively short, indicating that the differences between the words used in the requirements from 1996 and the words used the subsequent years are limited.

Table 11. Complete list of keywords that are significantly more common in one set or the other.

N	More common 1997 than 1996	More common 1996 than 1997
1	SYMBOL	
2	CMICRO	
3	PAGE	
4	HMSC*	
5	TEXT	
6	SHOULD	
7	FLOW	
8	MSC	
9		ENV
10		ITEX
11		SEND
12		SPEC
13		N

* No occurence 1996

Johan Natt och Dag, Dept. of Communication Systems, Lund University, johan.nattochdag@telecom.lth.se
Kerstin Lindmark, Dept. of Linguistics, Lund University, kerstin.lindmark@ling.lu.se

Table 12. Complete list of keywords that are significantly more common in one set or the others.

N	More common 1996 than other years	More common other years than 1996
1	MINISYSTEM*	
2	MACRO	
3	N	
4	**LNO***	
5	AB	
6	SPEC	
7	CALLER*	
8	SDL	
9	SEND	
10	SHALL	
11	ANSWER	
12	**INMPOVEMENT***	
13	ACTIVEX*	
14	ROOT	
15	SDT	
16	TOOLBARS*	
17	RECEIVE	
18	**AR***	
19	**THANK***	
20	TIMER	
21	AWARE	
22	GATES	
23	COMMAND	
24	DEPENDENCY	
25	OOA	
26	SIMUI	
27		TELELOGIC
28		H
29		FILES
30		TEXT
31		PAGE
32		TAU
33		SYMBOL

* No occurence other years

6. Conclusions

In this paper we have investigated the possibility of constructing a domain-specific lexicon more or less automatically. Based on the analysis in Section 5, it may be concluded that this is possible if using the appropriate language source. Three language sources were compared, software requirements, software documentation, and the BNC Sampler. It was found that:

1. The language used in requirements may be considered to be unique compared to general spoken and written language.
2. The requirements and the documentation share basically the same domain-specific words, whereas the most used phrases differ.
3. The language in the requirements is fairly consistent over the years.

Thus, it may be concluded that:

1. The most appropriate language base for automatically constructing a lexicon would primarily be existing requirements.
2. An initial subset of requirements may be adequate to cover the language used when specifying requirements.

Further investigation on how the lexicon may be constructed is now of great interest. It is suggested that this may be aided by automated taggers, enabling a classification of terms and further automated, yet relevant, term extraction.

7. References

Carlshamre, P. (2002). *A Usability Perspective on Requirements Engineering – From Methodology to Product Development* (Dissertation No. 726). Linköping: Linköping University, Linköping Studies in Science and Technology.

Burnard, L. (Ed.) (2000). *The British National Corpus Users Reference Guide*. Oxford: Oxford University Computing Services, British National Corpus.

Dunning, T. (1993). Accurate Methods for the Satistics of Surprise and Coincidence. *Computational Linguistics, 19*, 61–74.

Kilgarriff, A. & Salkie, R. (1997). Using Word Frequency Lists to Measure Corpus Homogeneity and Similarity between Corpora. In *Proceedings of the Fifth ACL Workshop on Very Large Corpora*. New Brunswick, NJ: Association for Computational Lingustics.

Lubars, M., Potts, C., & Richter, C. (1993). A review of the state of the practice in requirements modelling. In *Proceedings of IEEE International Symposium on Requirements Engineering* (pp. 2–14). Los Alamitos, CA: IEEE Computer Society Press.

Natt och Dag, J., Regnell, B., Carlshamre, P., Andersson, M., & Karlsson, J. (2002). A feasibility study of automated natural

language requirements analysis in market-driven development. *Requirements Engineering, 7*, 20–33 .

Potts, C. (1995). Invented Requirements and Imagined Customers: Requirements Engineering for Off-the-Shelf Software. In *Proceedings of the Second IEEE International Symposium on Requirements Engineering* (pp. 128–130). Los Alamitos, CA: IEEE Computer Society Press.

Sawyer, P. (2000). Packaged Software: Challenges for RE. In *Proceedings of Sixth International Workshop on Requirements Engineering: Foundation for Software Quality* (pp. 137–142). Essen, Germany: Essener Informatik Beiträge.

Siegel, S., & Castellan, N. J (1988). *Nonparametric Statistics for the Behavioral Sciences* (2nd ed.). New York: McGraw-Hill.

Suonuuti, H. (2001). *Guide to Terminology* (2nd ed.). Helsinki, Finland: Tekniikan sanastokeskus.

Telelogic Tau 4.2 Documentation (2001). Malmö, Sweden: Telelogic AB.

Proceedings of the 2002 IEEE
Int. Workshop on Robot and Human Interactive Communication
Berlin, Germany, Sept. 25-27, 2002

A Prototype Robot Speech Interface with Multimodal Feedback

Mathias Haage[+], Susanne Schötz[×], Pierre Nugues[+]

[+]Dept. of Computer Science, Lund Institute of Technology,
SE-221 00 Lund, Sweden;
E-mail: Mathias.Haage@cs.lth.se, Pierre.Nugues@cs.lth.se

[×]Dept. of Linguistics, Lund University,
SE-221 00 Lund, Sweden;
E-mail: Susanne.Schotz@ling.lu.se

Abstract

Speech recognition is available on ordinary personal computers and is starting to appear in standard software applications. A known problem with speech interfaces is their integration into current graphical user interfaces. This paper reports on a prototype developed for studying integration of speech into graphical interfaces aimed towards programming of industrial robot arms. The aim of the prototype is to develop a speech system for designing robot trajectories that would fit well with current CAD paradigms.

1 Introduction

Industrial robot programming interfaces provide a challenging experimental context for researching integration issues on speech and graphical interfaces. Most programming issues are inherently abstract and therefore difficult to visualize and discuss, but robot programming revolves around the task of making a robot move in a desired manner. It is easy to visualize and discuss task accomplishments in terms of robot movements. At the same time robot programming is quite complex, requiring large feature-rich user interfaces to design a program, implying a high learning threshold and specialist competence. This is the kind of interface that would probably benefit the most from a multimodal approach.

This paper reports on a prototype speech user interface developed for studying multimodal user interfaces in the context of industrial robot programming [5]. The prototype is restricted to manipulator-oriented robot programming. It tries to enhance a dialogue, or a design tool, in a larger programming tool. This approach has several advantages:

- The speech vocabulary can be quite limited because the interface is concerned with a specific task.
- A complete system decoupled from existing programming tools may be developed to allow precise experiment control.
- It is feasible to integrate the system into an existing tool in order to test it in a live environment.

The aim of the prototype is to develop a speech system for designing robot trajectories that would fit well with current CAD paradigms. The prototype could later be integrated into CAD software as a plug-in.

Further motivation lies in the fact that current available speech interfaces seem to be capable of handling small vocabularies efficiently, with performance gradually decreasing as the size of the vocabulary increases. This makes it interesting to examine the impact of small domain-specific speech interfaces on larger user interface designs, perhaps having several different domains and collecting them in user interface dialogues.

The purpose of the prototype is to provide an experimental platform for investigating the usefulness of speech in robot programming tools. The high learning threshold and complexity of available programming tools makes it important to find means to increase usability. Speech offers a promising approach.

The paper is organized as follows: speech, multimodal interfaces, and robot programming tools are briefly recapitulated. Then, the prototype is described giving the design rationale, the system architecture, the different system parts, and a description of an example dialogue design. The paper concludes with a discussion of ongoing experiments and future enhancements to the prototype.

Figure 1: SAPI 5.1 speech interface application front end with a list of available command words.

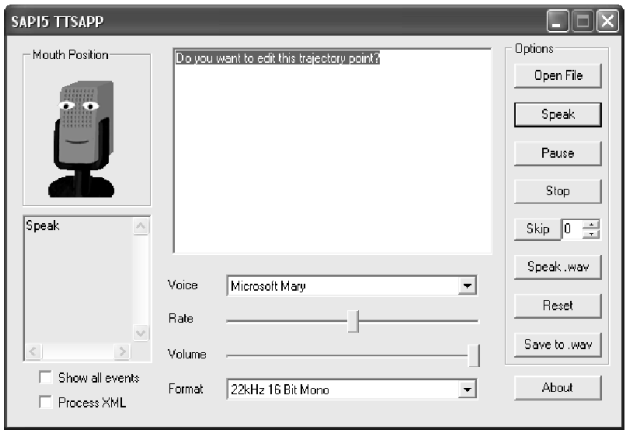

Figure 2: The SAPI 5.1 sample TTS application modified for use by the prototype system.

2 Speech, multimodal interfaces and robot programming tools

2.1 Speech recognition and synthesis

Speech software has two goals: trying to recognize words and sentences from voice or trying to synthesize voice from words and sentences. Most user interfaces involving speech need to both recognize spoken utterances and synthesize voice. Recognized words can be used directly for command & control, data entry, or document preparation. They can also serve as the input to natural language processing and dialogue systems. Voice synthesis provides feedback to the user. An example is the Swedish Automobile Registry service providing a telephone speech interface with recognition and synthesis allowing a user to query about a car owner knowing the car registration plate number.

A problem with speech interfaces is erroneous interpretations that must be dealt with [8]. One approach to deal with it is to use other modalities for fallback or early fault detection.

2.2 Multimodal user interfaces

A multimodal user interface makes use of several modalities in the same user interface. For instance, it is common to provide auditory feedback on operations in graphical user interfaces by playing small sounds marking important stages, such as the finish of a lenghty compilation in the Microsoft Visual C++ application. Rosenfeld gives an overview in [7].

Different modalities should complement each other in order to enhance the usability of the interface. Many graphical interfaces, including robot programming interfaces, are of the direct-manipulation type. Speech should therefore complement direct-manipulation interfaces [2]. Grasso [4] lists complementary strengths and weaknesses related to direct-manipulation and speech interfaces:

- Direct manipulation requires user interaction. It relies on direct engagement and simple actions.
- The graphical language used in direct manipulation interfaces demands consistent look and feel and no reference ambiguity to be usable. This makes it best suited for simple actions using visible and limited references.
- Speech interface is a passive form of communication. The medium allows for describing and executing complex actions using invisible and multiple references. It does not require use of eyes and hands making it suitable for hand-eye free operations.

Put in other words: speech might be used to avoid situations where you know exactly what you want to do but do not have a clue as where to find it in the graphical user interface. It may also help to avoid situations when you are able to describe an operation but do not know how it is expressed in the user interface.

2.3 Industrial robot programming interfaces

Essentially all robot programming boils down to the question of how to place a known point on the robot at a wanted position and orientation in space at a certain point in time.

For industrial robot arms, the known point is often referred to as the tool center point (TCP), which is the point where tools are attached to the robot. For instance, a robot arm might hold an arc-welding tool to join work pieces together through welding. Most robot programming tasks deal with the specification of paths for such trajectories [3].

Below is discussed how modeling of trajectories is performed in three different tool categories for programming industrial robots.

Teach pendant

A single robot operated by a person on the factory floor is normally programmed using a handheld terminal. The terminal is a quite versatile device. For instance, the ABB handheld terminal offers full programmability of the robot. The terminal has a joystick for manual control of the robot. Function buttons or pull-down menus in the terminal window give access to other features. Program editing is performed in a syntax-based editor using the same interface as for manual operation, i.e. all instructions and attributes are selected in menus. Special application support can be defined in a way uniform to the standard interface.

Trajectories are designed by either jogging the robot to desired positions and record them or by programming the robot in a programming language. For ABB robots the programming language used is called RAPID [1].

Off-line programming and simulation tools

In engineering environments, programming is typically performed using an off-line programming tool. An example is the Envision off-line programming and simulation tool available from Delmia. These tools usually contain: An integrated development environment. A simulation environment for running robot programs. A virtual world for visualizing running simulations and being used as a direct manipulation interface for specifying trajectories.

Trajectories are designed by programming them in a domain-specific language or by directly specifying points along the trajectory. The simulation environment provides extensive error checking capabilities.

CAD and task level programming tools

Task level programming tools typically auto-generate robot programs given CAD data and a specific task, for instance to weld ship sections. The software works by importing CAD data and automatically calculate necessary weld trajectories, assign weld tasks to robots and generate programs for these robots. These tools are typically used for programming large-scale manufacturing systems.

IDE	*Visualization*	*Programming*
Teach pendant	Real env.	Jogging & lang.
Off-line tool	Virtual env.	Lang. & sim.
Task-level tool	Virtual env.	CAD data

Table 1: Visualization and programming in different categories of robot programming tools.

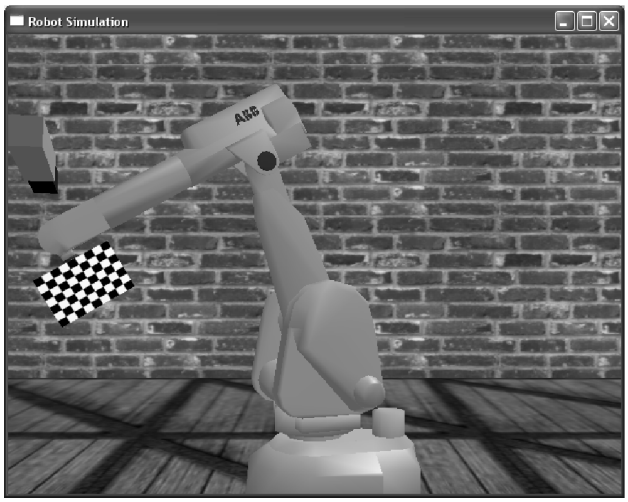

Figure 3: Virtual ABB IRB 2000 industrial robot arm with 6 degrees of freedom (developed in cooperation with Tomas Olsson, Dept. of Automatic Control, Lund University, email: tomas.olsson@control.lth.se).

3 Prototype

Two observations can be made concerning the user interfaces in the above programming environments: The typical task performed by all IDEs (Integrated Development Environment) is to model task specific robot trajectories, which is done with more or less automation, depending on tool category. The user interface consists of a visualization and a programming part, see Table 1.

The prototype presented here is a user interface where speech has been chosen to be the primary interaction modality but is used in the presence of several feedback modalities. Available feedback modalities are text, speech synthesis and 3D graphics.

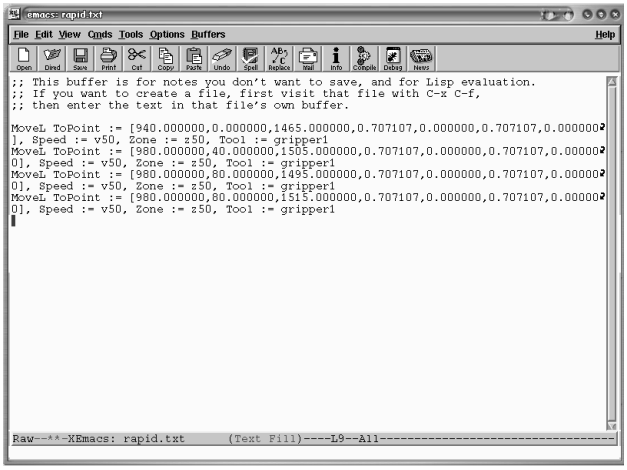

Figure 4: XEmacs is used as trajectory editor and database.

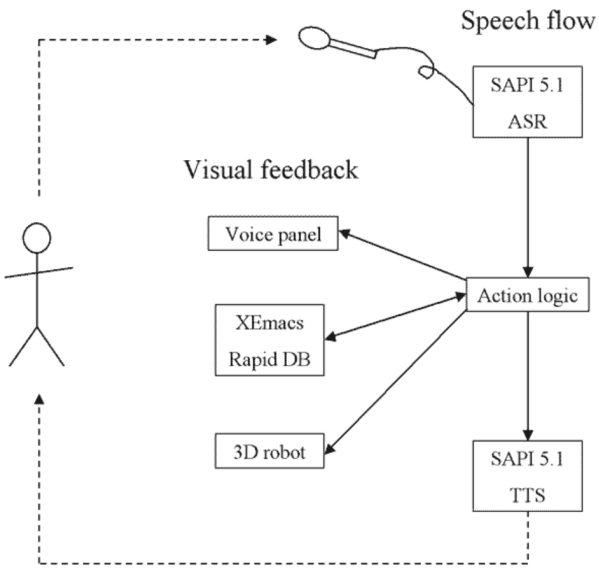

Figure 5: Prototype system dataflow.

The prototype system utilizes the speech recognition available in the Microsoft Speech API 5.1 software development kit. The SAPI can work in two modes: command mode recognizing limited vocabularies and dictation mode recognizing a large set of words and using statistical word phrase corrections. The prototype uses the command mode. It is thus able to recognize isolated words or short phrases [6].

The system architecture uses several applications (see Figures 1, 2, 3, 4): The *Automated Speech Recognition* application, which uses SAPI 5.1 to recognize a limited domain of spoken user commands. Visual feedback is provided in the Voice Panel window with available voice commands. The *Action Logic application*, which controls the user interface system dataflow and is the heart of the prototype. The *Text-To-Speech* application synthesizing user voice feedback. The *XEmacs* application acting as a database of RAPID commands and also allowing keyboard editing of RAPID programs. The *3D Robot* application providing a visualization of the robot equipment.

A decision was made to not use any existing CAD programming system in the prototype. The reasons were twofold: extending an existing system would limit the interaction into what the system allowed, making it difficult to easily adjust parameters like the appearance of the 3D world and the behavior of the editor. The second reason is that by not including a commercial programming system it is possible to release this prototype into the open source community as a complete system.

3.1 System architecture

The prototype system architecture follows a traditional client-server approach. The action logic application acts as a server with all other applications acting as clients. Interprocess communication is performed using Microsoft Win32 named pipes and sockets.

The system dataflow is centered around the speech applications since it is the primary modality of the system. Basically information flows from the speech TTS to speech synthesis application through the action logic application. The action logic application then interacts with the other applications (XEmacs, 3D robot) in order to update the state and different views supported in the interface (Figure 5).

3.2 Prototype applications

Action Logic

The action logic application is the heart of the system. All information goes through this application. The logic controlling the interface is hidden here.

The basic work flow of the application is:

1. Receive spoken commands from the speech recognition application.

2. Interpret the commands and act accordingly: *Send Lisp editing commands* to the XEmacs editor that is storing the trajectory as a sequence of RAPID MoveL (Move Linear) commands. *Read trajectory* stored in XEmacs and send it to the 3D

application for execution and visualization. *Send feedback* to be spoken to the speech synthesis application.

Microsoft SAPI 5.1 speech recognition and synthesis

The speech recognition and synthesis applications are based on the Microsoft Speech API version 5.1. Each application is built by utilizing an example application delivered together with the SDK and modifying it for our purposes. The example applications used for the prototype are CoffeeS0 and TTSApp.

The modifications necessary were quite small. They included: Adding communication capabilities to the applications so that they could send and receive information from the action logic application. This was done by adding a new communication thread to the application. Modifying the application window message handler to issue and receive speech messages from the new communication code. Changing the user interface to show our domain-specific vocabulary. And finally tune the speech recognition application to our vocabulary. This was done by rewriting the default XML grammar read into the speech recognition application upon initialization.

XEmacs RAPID trajectory editing and database

XEmacs is utilized as a combined database, editing and text visualization tool. The trajectory being edited is stored in an XEmacs buffer in the form of a sequence of RAPID MoveL commands:

```
MoveL ToPoint := [940,0,1465,0.707,0,0.707,0],
   Speed := v50, Zone := z50, Tool := gripper1
MoveL ToPoint := [980,80,1495,0.707,0,0.707,0],
   Speed := v50, Zone := z50, Tool := gripper1
```

The trajectory code is visualized in text form in the XEmacs buffer window. It may be edited using normal XEmacs commands. Thus the interface, even if developed with speech in focus, allows alternate interaction forms.

The interaction between XEmacs and the action logic application is done using LISP, see Table 2. The action logic application phrases database insertion/removal/modification commands of trajectory parts as buffer editing commands. These are executed as batch jobs on the XEmacs editor using the gnuserv and gnuclient package.

Spoken command	*Emacs LISP*
Add point	(kill-new "MoveL..."), (yank)
Remove point	(kill-entire-line)
Move forward	(forward-line 1)
Move backward	(forward-line -1)

Table 2: Sample LISP editing command sent to the Emacs RAPID database in response to spoken commands.

Virtual environment

The prototype needed a replacement for the 3D visualization usually shipped with robot programming applications to be realistic. A small 3D viewer previously developed was taken and enhanced with interpretation and simulation capabilities for a small subset of the RAPID language.

The tool is capable of acting as a player of trajectories stored in the XEmacs database. Player commands (play, reverse, stop, pause) is controlled from the action logic application.

3.3 Dialogue design

A preliminary experiment based on Wizard-of-Oz data obtained from the authors has been implemented.

The basic idea of this interface is to view trajectory modeling as editing a movie. It is possible to play the trajectory on the 3D visualizer, insert new trajectory segments at the current point on the trajectory, remove trajectory segments, and moving along the trajectory backward and forward using different speeds.

All editing is controlled using spoken commands, see Table 3. The user gets feedback in the form of a synthesized voice repeating the last issued command, seeing the trajectory in text form in the XEmacs buffer window and seeing the trajectory being executed in the 3D window. The command is always repeated by a synthesized voice in order to detect erroneous interpretations immediately. At some points (for critical operations like removal of trajectory segments), the interface asks the user if he/she wants to complete the operation.

4 Ongoing experiments and future work

The prototype will be used to explore the design space of speech interfaces with multimodal feedback. Below follows a few issues that would be interesting to gather data on:

- Varying the degree of voice feedback, as well as the type of information conveyed.

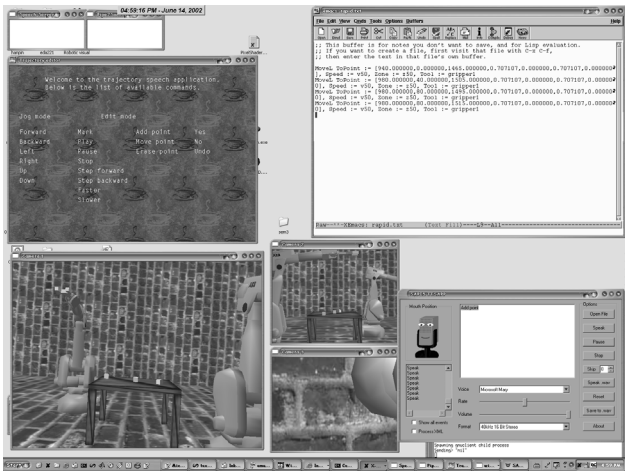

Figure 6: The prototype system user interface consists of four windows; 1. The voice panel containing lists of available voice commands. 2. The XEmacs editor containing the RAPID program statements. 3. The 3D visualization showing the current state of the hardware. 4. The TTS application showing the spoken text.

- Varying between different kinds of visual feedback.
- Varying the command vocabulary and interface functionality. For instance by allowing some task level abstractions in movement specifications, i.e. move to object, grab object.

For the future, there is a list of wanted extensions:

- Allow multiple domains in the speech recognition application, with the option of choosing which one to be applied from the action logic application. This feature could be used to test speech interfaces with state.
- Allow the entire experiment interface configuration to be specified in XML. Remove all hacking necessary to tune the interface. This would also speed up development since it would be easy to switch between different configurations.

5 Conclusion

We have developed a speech interface to edit robot trajectories. An architecture based on reusable application modules was proposed and implemented.

The work is aimed at studying feasability and usefulness of adding a speech component to existing software for programming robots. Initial feedback from users of the interface are encouraging. The users, including the authors, almost immediately wanted to raise the abstraction level of the interface by referring to objects in the surrounding virtual environment. This suggests that a future interface enhancement in such direction could be fruitful.

Spoken commands	*Purpose*
Forward, backward, left, right, up, down	Jog robot
Play, stop, step forward, step backward, faster, slower	Play trajectory
Mark, add point, move point, erase point	Edit trajectory
Yes, no	User response
Undo	Undo

Table 3: Vocabulary used in the prototype.

References

[1] ABB Flexible Automation, S-72168 Västerås, Sweden. *RAPID Reference Manual.* Art. No. 3HAC 7783-1.

[2] Cohen, Philip R. *The Role of Natural Language in a Multimodal Interface.* UIST'92 Conference Proceedings. Monterey, California. p. 143-149. 1992.

[3] Craig, John J. *Introduction to Robotics.* Addison-Wesley Publishing Company. Reading, Massachusetts. 1989.

[4] Grasso, Michael A, Ebert, David S, Finin, Timothy W. *The Integrality of Speech in Multimodal Interfaces.* ACM Transactions on Computer-Human Intraction, Vol 5, No 4. p. 303-325. 1998.

[5] *Prototype homepage*, `http://www.cs.lth.se/~mathias/speech/`.

[6] *Microsoft Speech Technologies*, `http://www.microsoft.com/speech/`.

[7] Rosenfeld, Ronald, Olsen, Dan, Rudnicky, Alex. *Universal Speech Interfaces.* Interactions November + December. p. 34-44. 2001.

[8] Suhm, B., Myers, B., Waibel, A. *Multimodal Error Correction for Speech User Interfaces.* ACM Transactions on Computer-Human Interaction, Vol. 8, No. 1. p. 60-98. 2001.

LUNDS UNIVERSITET

Institutionen för Datavetenskap

http://www.cs.lth.se

Produktion: Jonas Wisbrant • 2004