Institut für Maschinelle Sprachverarbeitung
Abteilung Theoretische Computerlinguistik
Universität Stuttgart
Azenbergstr. 12
D–70569 Stuttgart

Grupo de Validación de Aplicaciones Industriales
Departamento de Inteligencia Artificial
Facultad de Informática
Universidad Politécnica de Madrid
28660 Boadilla del Monte, Madrid, España

Studienarbeit / Sistemas Informáticos Nr. 2177

A Multilingual Search Engine Based on the Universal Networking Language

Stefanie Wiltrud Kessler

Course of Study:	Computer Science
Examiner:	Prof. Ph.D. Hinrich Schütze
Supervisor:	Prof. Jesús Cardeñosa Lera
Commenced:	June 17, 2008
Completed:	December 17, 2008
CR-Classification:	H.3.1, H.3.3, I.2.7

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

The multilingual search engine presented here is part of a Patrilex project of the Validation of Industrial Applications (VAI) group with the Spanish Ministry of Culture (see 2.6 on page 14).

The ministry provides a collection of documents about the cultural heritage of Spain. These documents are in Spanish. To provide easier access for nonnative readers, the possibility to search in other languages should be added. It is assumed that these are people who will be able to understand the Spanish documents, but who might not know how to search for them, because they do not know the exact translation of the term they are searching. As the documents are of the specific domain cultural heritage, the desired search terms might not be in a general purpose dictionary. Hence the idea is to facilitate search for these users by allowing them to search in their own language, while still retrieving documents in Spanish. Currently the system supports the languages English, Arabic and Russian besides a direct search in Spanish.

The obvious way to retrieve documents using search terms in languages different from the indexed documents is to translate the search terms into the index language of the search engine. Another possibility is the translation of the documents to the query language, which is a much more resource intensive task on a large collection or with various input languages. And finally both search terms and documents might be translated to an intermediate language. This third approach offers interesting possibilities, as, for example, the easy integration of documents in different languages, but still poses the huge problem of translating all the documents to another language (see 2.1 on page 10). For this work, the first approach – translating the query to the index language – is followed.

The translation of the query to the index language is not done directly for every language pair, but via a common interlingua. This interlingua, the Universal Networking Language (UNL), is explained in more detail in chapter 2 on page 9. Using an interlingua greatly facilitates the addition of a new language to the system or the addition of documents in different languages to the index. It also leaves open the possibility to translate the documents to UNL one day and search independent of document language.

In order to translate, a dictionary of the input language and UNL has to be available. The dictionary contains the words of the input language as lemmata. To be able to search in the dictionary, the query has to be lemmatized. That means for every word the ”canonic” form has to be found. This normally is the singular form for nouns or the infinitive for verbs. The rules for lemmatization for the different languages are described in detail in chapter 3 on page 17.

Apart from the query translation an additional query expansion is done. Query expansion adds related words to the query. This method helps to find documents that do not contain the exact original search words, but are also relevant to the query. Related words are added to the query string based on a thesaurus. The creation of this thesaurus was also part of the Patrilex project. The thesaurus is in UNL.The query expansion is explained in detail in chapter 4 on page 25.

For the implementation, the Apache Lucene search engine framework is used, which is described along with basics of information retrieval in chapter 5 on page 29.

Details about the implementation can be found in chapter 6 on page 35. This includes a conceptual design of the multilingual search engine.

An evaluation and validation of the system was done in chapter 7 on page 41. the evaluation is centered mostly in evaluating the different possibilities of query expansion.

Words that are written in *italic* are usually examples.

Chapter 2

The Universal Networking Language (UNL)

This chapter is a short introduction to machine translation and the Universal Networking Language (UNL). Machine Translation is the process of translating a text from a source language to a target language by a computer. There has been a lot of research in this area from the 1950s on. Machine translation is one of the most fascinating problems of computational linguistics, but as many fascinating problems it is difficult to solve.

On word level we have words with different meanings depending on context or pronounciation. An example is *Time flies like an arrow*, where *flies* could be a noun or a verb and *like* could be a verb or an adjective. We also have phrases that are not to be translated literally as, for example, the Spanish *hacer cola* should not be translated literally as *make queue*, but as *to queue up*.

On sentence level we have ambiguous structures like the famous *He saw the man with the telescope*, where it is not clear if the observer saw the man with the help of a telescope or the observer saw a man who had a telescope. We also have to cope with anaphors (*They saw the Alps as they flew over Zurich* - what does the second *they* refer to?) and similar structures where only world knowledge allows us as humans to understand the sentences. But to encode this world knowledge in a way that helps translation is very difficult, it may even be impossible.

In the following chapter the rule-based approaches to machine translation are briefly discussed. There is a short introduction to UNL followed by a discussion of some strengths and weaknesses of UNL. And finally the Patrilex project is explained.

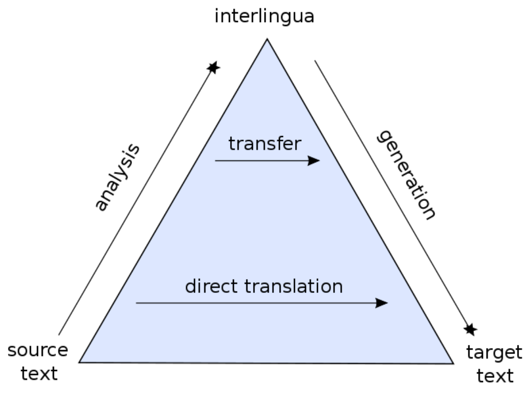


Figure 2.1: Levels of Machine Translation

2.1 Machine Translation and Interlingua

There are different methods in machine translation. But basically, the source text has to be analyzed and transferred into a formal representation, from which a text in the target language is generated. The differences between the traditional methods are where to stop the analysis. This is visualised in figure 2.1 [24].

As we can see, direct translation performs only a very shallow analysis (for example only morphology) and then directly translates the sentences to the target language. In the target language there may be some simple postprocessing like word reordering, but no further structural changes are done. As can be expected, this approach works well for related languages (like Spanish and Italian for example), but gives insufficient results for languages that differ substantially (like Arabic and Spanish).

Another approach that is used often in machine translation is transfer. There, the meaning of a text in the source language is transfered to a target language using a set of transfer rules. To achieve this, as a first step the text in the source language is analyzed. This analysis can stay at a syntactical level or go up to semantical level. The resulting representation is changed to a representation in the target language by a transfer module. From this representation (syntactic or else) a target language text is generated.

Transfer systems can produce high quality translations, but the disadvantage is obvious. For every pair of languages there has to be a transfer module with a set of (manually written) rules. A transfer-based system with N languages therefore needs $N \cdot (N-1)$ transfer modules (plus the analysis and generation modules).

The last one of the traditional aproaches is translation based interlingua. There, translation is performed by translating a text in a source language into an ″interlingua″ (sometimes also called ″pivot language″) and from that interlingua to the target language. So for each language in the system, only one module for translating text into the interlingua and one for translating from the interlingua has to be written. A system with N languages then has $2 \cdot N$ modules – far fewer than with transfer! The disadvantage of an interlingua is that the effort of analysis and generation is much bigger than in transfer systems.

The biggest challenge in interlingua systems is the design of the interlingua. An interlingua should be unambiguous. But it needs to be as expressive as all natural languages, so that every natural language can be translated into the interlingua without loss of meaning or important grammatical information (such as subject, tense, …). The interlingua can be a natural language, but usually an artificial language is used.

Additionally, there is the general problem that it is not feasible to capture all of the meaning of a text in natural language by any formalism. This problem is discussed in more depth by Gallardo in [12].

In the search engine that is described here the artificial language UNL is used as an interlingua throughout the system.

2.2 History of UNL

The UNL program was created by the United Nations in 1996 as a project of the University of United Nations [17]. Its goal was to overcome the language barriers on the internet by a machine-readable interlingua. The project included the development of a system to convert this interlingua to each of the participating languages. Initially, there were 15 participating languages: Arabic, Chinese, English, French, German, Hindi, Indonesian, Italian, Japanese, Latvian, Mongolian, Portugese, Russian, Spanish and Thai.

In May 2000, the Universal Networking Digital Language Foundation (UNDL Foundation) with several language centers was created. There is supposed to be one center for each language that would maintain the UNL language (the specifications, the dictionaries and the knowledge base), promote the UNL language and encourage the creation of centers for new languages.

Since 1997 the group of Validation of Industrial Applications (VAI)[1] of the Universidad Politécnica de Madrid is participating in the United Nations project and it is in charge of the Spanish language center [2].

2.3 Structure of UNL

In UNL a sentence is represented as a hypergraph, whose vertexes represent UNL concepts and the edges represent relations between the concepts. For a formal specification see [21]. Short explanations are given by Cardeñosa [17] and by Texeira and Nunes [20].

[1]`http://www.vai.dia.fi.upm.es`
[2]`http://www.unl.fi.upm.es`

The UNL concepts are not created as abstract concepts, but rather as word meanings. They are represented by Universal Words (UWs). Universal Words consist of a Head Word and a list of constraints. The Head Word normally is one English word, but it may consist of various words (like *abide by*). If there is no English word for a concept, a word in another language may be used (like *samba*).

The constraint list may be empty if the word is a basic concept that doesn't need disambiguation (like *banjo* or *bank_robber*). But if a word is ambiguous, the disambiguation takes place in the restrictions. The restrictions are relations of this Universal Word with other Universal Words. Often the relations *icl* (kind of) or *equ* (equal to) are used. This reflects for example the difference between the bank where one leaves money (*bank(icl>financial_institution>thing, equ>depository_financial_institution)*) and the bank of the river (*bank(icl>slope>thing)*).

Universal Words are classified as nominal, verbal, adjective or adverbial concepts. Compound universal words can be constructed to express compound concepts like in *[Women who wear big hats in movie theaters] should be asked to leave* [17].

Two Universal Words can have a relation with each other. There are a number of defined relations. Relations try to cover ontological relations (like synonymy), logical relations (like disjunction) and thematic roles (like agent). The set of predefined relations is supposed to be fixed.

There are also attributes that can be given to a Universal Word. These attributes contain grammatical information (like number, time), attitudes (like interrogative, polite), focus (topic, emphasis) and points-of-view (need, will, expectation). They are written behind the Universal Word, connected with an @. For example *potato(icl>food)@pl* means *potatoes.*

Every sentence has an entry-node that is marked with the attribute *@entry*. It denotes the main node of a UNL expression.

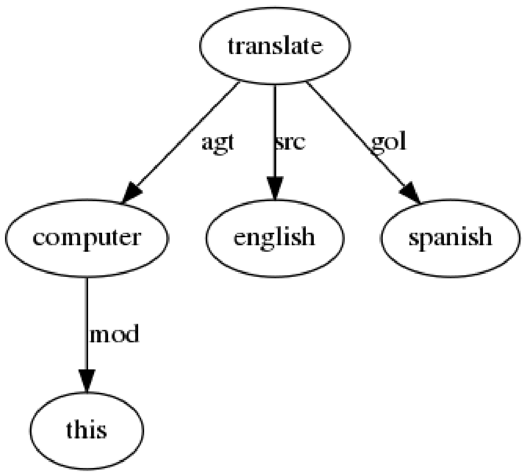


Figure 2.2: UNL Graph of the Sentence *This computer translates from English to Spanish*

Figure 2.3 on the preceding page shows the structure of the UNL graph for the sentence *This computer translates from English to Spanish*. The UNL concept *computer* is the agent of the action, the source is the language *English* and the goal the language *Spanish*. For simplicity, restrictions have been omitted.

2.4 Applications of UNL

UNL was created to provide a tool for making web content available in different languages. The original idea is to translate the pages to UNL with an EnConverter, then store them in UNL and create a version in the needed language with a DeConverter on demand. The process of enconverting is interactive, the user has the possibility to change his sentences in the original language and in the resulting UNL graph. There are implementations of En- and DeConverters in a variety of languages available (see for example [26] for French, [22] for Chinese).

The HEREIN System, result of the HEREIN project of 1999, is an internet-based facility for improving cultural heritage management methods in Europe [3]. There are about 30 countries with 24 languages involved. Every country delivered a report in its language about cultural heritage issues. The costs for translating every report to every language (by a human translator!) would be very high. The big advantage of UNL is, that if a correct UNL graph structure of the document is created once, it can be translated to every language where a deconverter exists.

As a last example, there is the attempt to improve case based retrieval of a case based reasoning system by using UNL [16]. Case based reasoning is widely used for problem solving. It is based on a large knowledge base that contains problem descriptions and their solutions. On the input of a problem, the most similar cases in the knowledge base are returned to the user as a solution. An existing system on the domain of software problems was to be improved using a problem description in natural language as an input. Both the problem and the titles of the cases in the knowledge base were then mapped to UNL concepts. To measure the similarity of a problem and the cases, the distance in a hierarchy between the concepts of the problem and the cases was computed. Also the structure of the UNL graphs was compared using the number of common subgraphs as a measure. The system was able to return cases as *I am unable to browse the Internet* for the problem *I am unable to surf the web*, a match that such systems that only compare strings cannot manage due to the difference in word choice (*browse* and *Internet* compared to *surf* and *web*).

2.5 Advantages and Limitations of UNL

One advantage of UNL is that it is based on a natural language (English). Therefore it has all the expressiveness in vocabulary of that language. Because it is possible to add words from other languages as headwords for new concepts, UNL even has a greater expressiveness in vocabulary. With the restrictions of the UWs, the ambiguity of the English words is eliminated [12].

English is the basis for UNL, but UNL is not only an extended form of English. You could look at the headword to be just any random string chosen as an identifier for the word meaning. With the help of the UNL Knowledge Base (UNLKB), the meaning of the word should be defined by its relations [3].

UNL sentences are independent from the source language. After getting the UNL expression for a sentence, we no longer have to store the source language sentence. All information is contained in the UNL graph.

UNL is readable by people with little training, provided that the person speaks English, which today is true for many people. Many problems in converting text in a given language to UNL can be solved by a human interactively.

One of the limitations of UNL is, that it is developed by a lot of different people from different backgrounds in different countries. This leads to a considerable diversity between the UNL terms of the different centers. If the trend continues, there will be different UNL dialects that might need translation from one to the other [1]. The same problem is also reflected in occasionally inconsistent hierarchies in the knowledge base. Some relations of UNL are difficult to distinguish or to assign which will lead to different but equally correct results for the same sentence if translated by different persons.

UNL mixes semantics and syntax on some occasions. There are different UWs for the word *good* in its predicative and attributive use. While there is a difference in English, it is doubtful that this holds true for every language. Even more problematic is the classification as nominal, verbal, adjective or adverbial. Words that are used in a nominal way in one language might be translated differently in another language. Texeira and Nunes [20] bring as an example *She enjoys reading books*, where *reading(icl>action)* is a nominal concept in UNL, but would be translated into Portuguese as *Ela gosta de ler livros*, where *ler* is a verb.

As a conclusion, one might say that UNL suffers from the same limitations as all interlinguas, but offers some advantages that make it useful for certain applications. In the case of my multilingual search engine, the advantages outweigh the problems, because as we only translate single words the difficulties of defining relations do not affect translation. Also, as long as the UWs are consistent in the dictionaries for the different languages, dialects or problematic restrictions do not pose problems.

2.6 The Patrilex Project

The Patrilex project is a project of the VAI group with the Spanish Ministry of Culture. The objective of this project is the development of a methodology to support the creation of multilingual lexical resources for information retrieval applications in multilingual environments [18]. In the process, two tools (TesaurVAI and this search engine) and several resources were created. The domain of the project is cultural heritage. The project started in 2005 and was to end in December 2008, but will most likely be extended until summer 2009.

The project is composed of several tasks:

- Definition of a methodology for creating multilingual lexical resources
- Creation of a multilingual search engine
- Automatisation of extraction of terms of a domain and the management of the resulting thesaurus (TesaurVAI)
- Creation of multilingual resources (dictionaries and a thesaurus)

One of the tasks of the project is the development of a search engine with a document repository in various languages, where it is possible to search in various languages. Due to the fact that the documents provided on the domain where exclusively in Spanish, the search engine presented here works only on Spanish documents.

In the project, several resources have been created. One is a thesaurus of the domain of around 1300 words. More details are given in section 4.1 on page 25. Also dictionaries with translations of the words of the thesaurus into the supported languages have been created. In the search engine the thesaurus and the dictionaries are used.

The project was planned for English, Spanish and Russian. One central goal of the project is to assure the extendability to other languages and to develop a methodology for the easy incorporation of a new language to the system. In the course of the project, Arabic was added as a language and German will soon be added as well.

The key element for the easy addition of a dictionary to the project is the use of UNL as an interlingua. It is not necessary to translate from every language to the new language. This would be a lot of work and also the task of finding a translator for a "rare" pair (starting with Arabic - Russian) would be difficult. By using UNL, the only translation to be made by somebody is from UNL to the new language.

Also part of the project is the evaluation of whether the translation from UNL to another language is more or less difficult than the translation from a natural language (in the project it is compared with the translation from Spanish) and whether or not more errors were done. To make this comparision, the list of words was translated twice to every language, once from Spanish and once from UNL by a different person. The resulting translations will then be compared with regard to time needed and amount of errors made. The results will be contained in the final report of the project.

The search engine developed in the project is to be integrated into the home page of the Spanish Ministry of Culture.

Chapter 3

Language Processing

In almost all information retrieval applications there is some language pre-processing of the query done before searching for the words in the index. Most of the time this is stemming, the removal of suffixes to approximate a base form. This base form is not necessary a word form that actually exists. Often words of different grammatical categories are reduced to one form (like for example *prove* and *provable* might become *prov*). In contrast to this, in lemmatization it is attempted to find a canonical basic form of a word like the singular form of a noun in plural or the infinitive of a verb. These basic forms are always existing forms of the same grammatical category.

For the tasks at hand, the translation of the input words from the query, we cannot reduce different classes to one form without introducing unnecessary ambiguity in translation. Also for morphologically complex languages as Russian and Arabic stemming is difficult to realize. Still we have to do some pre-processing, because the dictionary is not a full-form dictionary, but contains lemmata. So we have to find the lemma for each query term before we can translate it using lemmatization. For every language there is a set of rules for lemmatization.

There is no tagger in the system that would annotate the category of the word, so it is always assumed that the words are nouns. This will, of course, lead to a lot of misinterpretations. However, since the users of search engines are mostly used to typing short queries consisting of two to three words (see [19], chapter 19), probably mostly nouns, the error might not be so great in the average usage. Also, the dictionary of the domain consists only of nouns of the domain, accompanied mostly by adjectives.

The language processing in the analysis module can be divided into three parts. First, tokenization, where the query string is split up into the different tokens. Then a pre-processing step, where the tokens are normalized. This is dependent on the language, but nearly always involves lowercasing. As a last step, the actual lemmatization is performed. For efficiency reasons, tokenizing and pre-processing are implemented as one step, although conceptually they are two.

Determiners:	el, la, lo, las, los, un, una, uno, unas, unos
Adjectives :	algún, alguno, alguna, algunos, algunas
	todo, cada, otro
ser:	ser, soy, eres, es, somos, sois, son, siendo
	fui, fuiste, fue, fuimos, fuisteis, fueron
estar:	estar, estoy, estás, está, estamos, estáis, están, estado, estaba
hacer:	hacer, hago, haces, hace, hacemos, haceis, hacen
haber:	haber, he, has, ha, hemos, habeis, han
ir:	ir, voy, vas, va, vamos, vais, van
Personal pronouns:	yo, tú, él, ella, nosotros, vosotros, ellos, ellas
Prepositions:	en, de, por, para, con, sin
Other:	pero, si, y, o, sí, no

Table 3.1: List of Spanish Stop Words

3.1 Tokenization

Tokenization is nearly the same for all languages in the system. As a general rule tokens are split at non-letter characters.

Stopwords are removed from the token list. Stopwords are the first few most common words in a language. By not including these words in the search index a lot of space can be saved ([19], chapter 2). As these words are of little importance to distinguish the words of the domain and since in the end only the words of the domain will be translated, the decision to remove stop words is quite natural.

In table 3.1 we see a list of Spanish stopwords as an example. The words that are removed are mainly articles, personal pronouns, forms of the word *to be*, and other verbs used for building composed past tense and future.

3.2 English

English is an Indo-European language that uses the Latin alphabet. There is no grammatical gender, nor is there declension of nouns. Nouns may be singular or plural. So for lemmatization, only words that are entered in plural have to be considered.

3.2.1 Normalization

The normalization for English consists of changing all letters of the tokens to lowercase. Also, accents are removed. Accents may occur on foreign words incorporated into English like *résumé*.

Singular ends in ...	Rule for plural	Implemented?
-um	change to -a	no
-on	change to -a	no
-a	append -e	yes
-us	change to -i	yes
-<vowel>x	change to -ices	yes
-sis	change to -ses	no
-xis	change to -xes	no

Table 3.2: List of English Plurals from Latin Origin

3.2.2 Lemmatization of Nouns

The default rule to form an English plural is to append an *s*. If the word ends with *x*, *ch*, *sh*, *ss*, *z* and sometimes *o*, there is an *es* added.

If the word ends with certain letters the final letter is changed. This is the case for words ending with a consonant followed by *y*. The *y* is changed in plural to *i* and *es* is appended. Also, nearly always if the word ends in *f* or *fe*, it is changed to *v* and *es* is appended.

Some English words from Latin origin form their plurals according to Latin plural forming rules. A list can be found in table 3.2.

We have to "invert" these rules to get from the plural form to the singular form. This creates the problem that sometimes there are various possible singular forms for one plural form. One problem is, for example, how to distinguish plurals with *ves* if they are from *f* or *fe*. In this case, the solution is that it is *fe* only if there is an *i* before the *f* (as in *knife*, but not in *wolf*) [15].

In other cases, where the rules overlap, the rule for words with Latin origin has not been implemented. This is the case for the rule changing *xis* to *xes* and the normal plural rule for words ending in *x*. In both cases the plural ends with *xes*. As there are more words ending with *x*, the other rule has not been implemented.

In addition, there are some words that regularly end with *s* and thus should not be lemmatized (like *bus*), or only occur in plural (like *scissors*), or have different meaning in singular and plural (like *goods*). These words are kept in a list, and before any other processing is done, the word to be lemmatized is checked to see if it is in the list and if that is the case it is not lemmatized.

And of course, we have the totally irregular plural. Fortunately, this is a very limited number of words and can also be managed in a list, where the corresponding singular is returned when a match is found. A list of the words can be found in table 3.3 on the next page.

Plural	men	firemen	women	children	oxen	feet	geese	teeth	lice	mice
Singular	man	fireman	woman	child	ox	foot	goose	tooth	louse	mouse

Table 3.3: List of English Irregular Plurals

3.3 Spanish

Spanish is also an Indo-European language. It uses the Latin alphabet. Spanish words can consist of all basic English letters (that is a-z) plus ñ and accented letters. Spanish has two genders, feminine and masculine. They are normally distinguished by their ending; female words normally end in *a* and masculine words in *o*, but words can also end in other letters. Spanish words form singular and plural. There is no declension. For lemmatization, once again we only have to consider plural.

3.3.1 Normalization

For Spanish the pre-processing consists of changing the case of all letters to lowercase and removing accents.

Accents in Spanish denote the syllable where the stress of the word is. There are very few words where the accent changes the meaning of the word (like *más* meaning *more* and *mas* meaning *but*). But by forming the plural, the accents can change because the stressed syllable is changed, and this change is quite difficult to reverse (*régimen* and *regimenes* versus *alemán* and *alemanes*). Because there is little information gained in keeping the accents and a bigger challenge to correctly assign them in lemmatization, the accented letters are exchanged for their non-accented equivalent before lemmatization begins.

3.3.2 Lemmatization of Nouns

Spanish plural rules are fairly simple. If the word ends with a vowel, an *s* is appended, if the word ends with a consonant, *es* is appended. If the last consonant is a *z*, it is changed to a *c*; final *g* becomes *gu* and final *c* becomes *qu* before adding *es*. Words ending with *í* or *ú* may build the plural with *s* or *s*. Foreign words normally build the plural only with *s* (like *chips*, *mamuts* or *cómics*, [8]).

So by reversing these rules, we have to look at the second-to-last letter. If it is anything but an *e*, we just delete the ending *s*. If it is an *e* the problem is how to know if the word ends with a consonant and an *es* was appended or if the word ends with *e* and an *s* was appended.

As there are no rules we opted for making a list of letters and two-letter combinations that Spanish words never end in (leaving aside foreign words or names). When there is one of these letters or letter combinations found, it is assumed that the word ends in an *e* and only the *s* is removed. This does not guarantee 100% accuracy, but it gives a decent result with the list of words at hand.

The letters and two-letter combinations Spanish words normally don't end in – and thus are nearly always followed by an *e* – are *b* (*nube*), *c* (*enlace*), *f* (*jefe*), *v* (*llave*), *p* (*golpe*), *ll* (*calle*), *j* (*viaje*), *ch* (*coche*), *t* (*aguacate*), *qu* (*parque*), *m* (*informe*), *l* and *r* preceded by a consonant (*mueble*) and *n* followed by a consonant (*estante*, *balance*) with the exception of *ing*.

Spanish words can end in *z* (*luz*), *y* (*ley*), *s* (*gas*), *n* (*elección*) or *r* (*flor*).

Unfortunately, there are also letters where it is equally possible to end with or without an *e*. These is for example *d* (*ciudad* or *alcalde*). In this case *es* is removed, but this decision is purely random.

Like in English, there are some words that regularly end with *s* and thus, should not be lemmatized (like *autobús*) or only occur in plural (like *tijeras*). They are treated as in English, stored as a list and not lemmatized.

3.3.3 Lemmatization of Adjectives

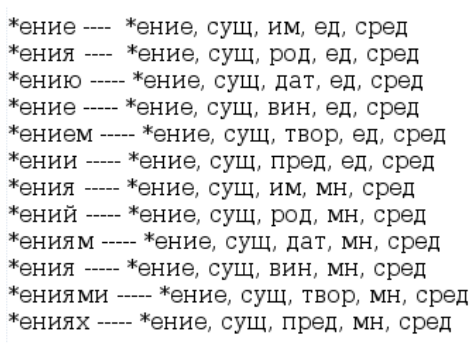
Spanish adjectives change their ending according to the noun they are associated with. The canonical form of the adjective is the male singular, like, for example, *rojo* (*red*). If the adjective is used with a female noun, the final *o* is changed to *a*, like in *una casa roja* (*a red house*). If the noun is in plural, a plural *s* is also added to the adjective.

Since the treatment of the plural is the same as for the nouns, male plural adjectives will be correctly lemmatized. But for adjectives in female form, we have the problem, that there is no tagging, so that we cannot know if the word is an adjective or not. We can't change the vowel in every case, because the changing of a vowel in a noun might result in a completely different word (*plaza* means place or square, but *plazo* means deadline).

At the moment, there is no treatment of these adjectives in the hopes that the users type them in the correct form – the same form as they appear in the dictionary and (supposedly) in the collection.

3.4 Russian

Russian is also an Indo-European language, but from the Slavic family. It has three genders, masculine, feminine and neuter. Words can occur in singular and plural and in any of the six cases (nominative, accusative, genitive, dative, prepositional and instrumental). It uses the Cyrillic alphabet. This section is written based on information of Igor Boguslavsky, in charge of the Russian dictionary and lemmatization.



```
*ение ----  *ение, сущ, им, ед, сред
*ения ----  *ение, сущ, род, ед, сред
*ению ----- *ение, сущ, дат, ед, сред
*ение ----- *ение, сущ, вин, ед, сред
*ением ----- *ение, сущ, твор, ед, сред
*ении ----- *ение, сущ, пред, ед, сред
*ения ----- *ение, сущ, им, мн, сред
*ений ----- *ение, сущ, род, мн, сред
*ениям ----- *ение, сущ, дат, мн, сред
*ения ----- *ение, сущ, вин, мн, сред
*ениями ----- *ение, сущ, твор, мн, сред
*ениях ----- *ение, сущ, пред, мн, сред
```

Figure 3.1: Example of a Russian Lemmatization Rule (Word Form - Lemma, Word Kind, Case, Number, Gender)

3.4.1 Normalization

The normalization for Russian consists only in changing all letters to lowercase.

3.4.2 Lemmatization of Nouns

Because we have to consider six cases, three genders and two numbers, the rules for Russian lemmatization are quite complex. Fortunately, they did not have to be written for this project, but were provided by Igor Boguslavsky, member of the Patrilex project. The rules are from the Laboratory of Computational Linguistics of the Institute of Information Transmission Problems of the Russian Academy of Science. They originally form part of a system for morphological analysis. The analysis is based on a morphological dictionary, but there are rules for guessing the form of new words that are not in the dictionary. These rules are the rules that have been implemented in the Russian lemmatizer. But they only contain cases where the decision is clear, so there will be a lot of cases where the rules are not sufficient. There are rules for about 80 types of lemmata.

An example of a rule for one set of word forms that have to be matched to the same lemma is shown in figure 3.1. The * represents any characters of the word. The ending of the word is then changed to the form on the right. After the lemma there is some grammatical information (word kind, case, number, gender), which is ignored for the lemmatizing.

3.5 Arabic

This section was written based on the information I got from Adriana Toni, who was in charge of writing the rules for the Arabic lemmatizer, and who also took part in the translation of the dictionary to Arabic.

Arabic is a Semitic language. It is written with Arabic letters from right to left. It has two genders, masculine and feminine. There are three cases, nominative, genetive and accusative and three numbers, singular, plural and dual. Arabic has a very structured morphology, but the rules are quite different from the rules of Indo-European languages.

3.5.1 Normalization

The Arabic letters are all consonants or long vowels. Short vowels are written above and below the consonants. In normal texts like newspapers they are not written at all, because it is clear from the context what vowels belong there. But sometimes they are written or partially written.

It is problematic to correctly add the vowels to a word. Also the information gained from a vowel is not that significant. For simplyfying the rules, and because the assumption is that Arabic-speaking people won't bother with vowels if they want to run a quick search, the vowels and diacritic symbols are deleted before lemmatization. Considered as Arabic short vowels are unicode code points U+064B through U+065E.

3.5.2 Lemmatization of Nouns

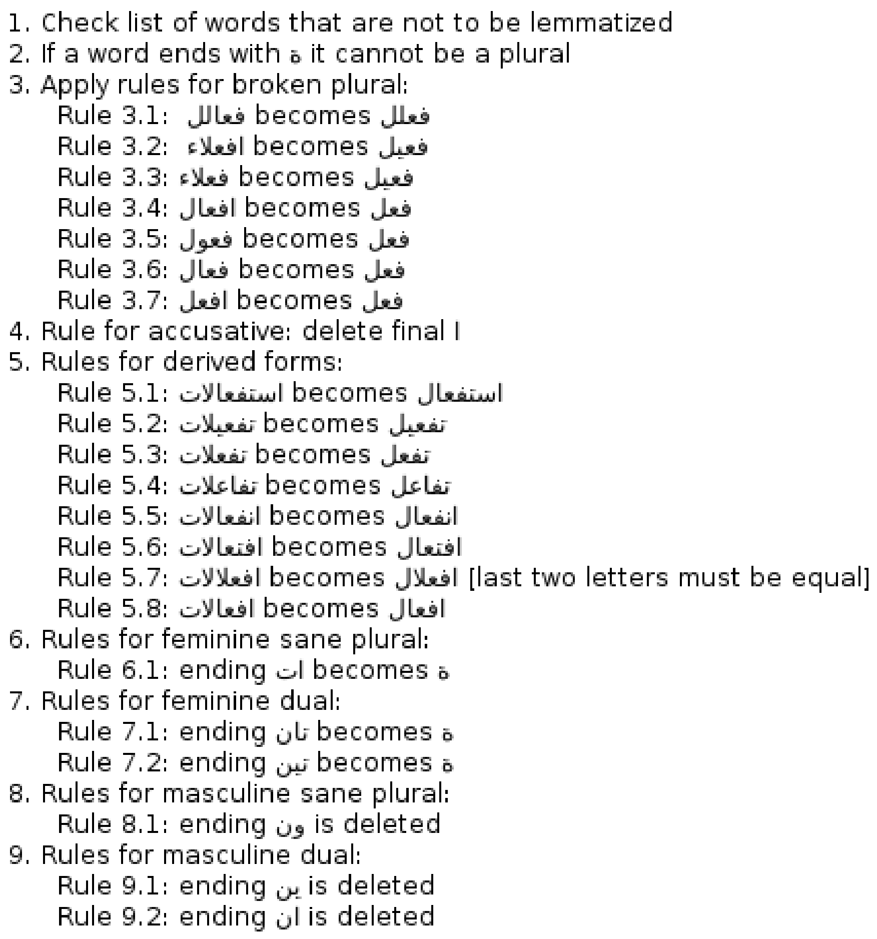
Feminine nouns usually end with the letter "ta marbuta" in singular, but there are exceptions; words that are feminine and don't end with ta marbuta and words that are masculine and end with ta marbuta. We don't have any possibility for determining the gender, so we ignore it.

There are three numbers in Arabic, singular, dual and plural. There are two types of plural in Arabic, the sane plural and the broken plural. The broken plural is formed by nouns that cannot form the sound plural. To form the plural form for a specific singular word that forms the broken plural is very difficult. But the reverse task, to find the singular form for a broken plural form, is possible without problems for nearly every broken plural form.

In addition there are also derived verb forms. They extend or modify the meaning of the root form. They are formed by adding letters before or between the radicals. For each of the derived verb forms, the rules for constructing the verbal nouns and participles are well defined, and we know what their plural form will be.

Figure 3.2 on the next page shows the rules that have been implemented and the order in which they have been implemented. The rules for broken plural and the derived verb forms are applied first, because if we detect one of these forms, we can be sure about the singular form. There are exceptions to the rules for regular dual or plural, words that, for example, look like a feminine sane plural, but are masculine. Exceptions are looked up in a list, but these lists are not very exhaustive at the moment.

For the rules for broken plural and the derived verb forms the pattern *fa - ayn - lam* is used. The letter *fa* represents the first letter of a word, *ayn* the second and *lam* the third. If there are four letters to a word (as in rule 3.1), there is a second *lam* but which must not be the same actual letter as the first letter represented by *lam*.



1. Check list of words that are not to be lemmatized
2. If a word ends with ة it cannot be a plural
3. Apply rules for broken plural:
    Rule 3.1: فعالل becomes فعلل
    Rule 3.2: افعلاء becomes فعيل
    Rule 3.3: فعلاء becomes فعيل
    Rule 3.4: افعال becomes فعل
    Rule 3.5: فعول becomes فعل
    Rule 3.6: فعال becomes فعل
    Rule 3.7: افعل becomes فعل
4. Rule for accusative: delete final ا
5. Rules for derived forms:
    Rule 5.1: استفعالات becomes استفعال
    Rule 5.2: تفعيلات becomes تفعيل
    Rule 5.3: تفعلات becomes تفعل
    Rule 5.4: تفاعلات becomes تفاعل
    Rule 5.5: انفعالات becomes انفعال
    Rule 5.6: افتعالات becomes افتعال
    Rule 5.7: افعلالات becomes افعلال [last two letters must be equal]
    Rule 5.8: افعالات becomes افعال
6. Rules for feminine sane plural:
    Rule 6.1: ending ات becomes ة
7. Rules for feminine dual:
    Rule 7.1: ending تان becomes ة
    Rule 7.2: ending تين becomes ة
8. Rules for masculine sane plural:
    Rule 8.1: ending ون is deleted
9. Rules for masculine dual:
    Rule 9.1: ending ين is deleted
    Rule 9.2: ending ان is deleted

Figure 3.2: Arabic Lemmatization Rules Implemented

Chapter 4

Query Expansion and the Thesaurus

Query expansion is "the process of reformulating a seed query to improve retrieval performance in information retrieval operations" [25]. That means, that "related" words are added to the words of the original query. The definition of "related" can vary, so there are a number of various ways to do query expansion.

The majority of the search engines that do query expansion do it in the following way: They do a retrieval run with the initial query, and take the top X documents returned as relevant documents for the query. Then the terms that distinguish these documents from the rest of the documents of the collection are extracted and used to expand the query. The results of retrieval with the expanded query are then presented to the user.

The search engine presented here does not follow this approach, although it does query expansion. Here, the query is expanded with the help of a thesaurus. This should provide terms that are closer related, because they were defined so by a human. It also allows to add terms to the query that do not appear in documents that are retrieved by the original query. The thesaurus was developed as a part of the Patrilex project.

The definition of a thesaurus is the "vocabulary of an indexation language formally organized with the goal of making explicit the relations between concepts." [7]. An indexation language here means a controlled set of natural language terms used to represent in a short form the topics of the documents. This means in our case, that the set of terms used to build the thesaurus are the most relevant terms of the collection.

Query expansion may increase recall, that means that it may help to find more of the documents of the collection that are relevant to the query. But it might harm precision, because we might expand with too many terms that don't have anything to do with the original term. This is especially relevant for ambiguous terms.

4.1 The Thesaurus

The thesaurus which is used in the multilingual search engine is in the domain of cultural heritage, and its developement was part of the Patrilex project.

Angeles Maldonado of the Centro de Información y Documentación Científica of the Consejo Superior de Investigaciones Científicas is responsible for the content of the thesaurus. She selected the terms that should be included in the thesaurus and defined the relations.

The thesaurus was developed using the tool TesaurVAI which was written to facilitate the management of thesauri in the departement as a part of the Patrilex project.

Relations that are defined in the thesaurus are (as defined by the Asociación Española de Normalización y Certificación [7]):

USE Denotes the term that is commonly used for denoting this concept (in case of synonyms or compound meanings)

TC (termino cabecera) This denotes the name of the broadest class the term belongs to (if the domain is divided in multiple subdomains)

UP (usado por) (Quasi-)synonyme of the term.

TG (término genérico) Broader term

TGG (término genérico genérico) Broader (more general) term (generic)

TGP (término genérico partitivo) Broader term (partitive)

TE (término específico) Narrower term

TEG (término específico genérico) Narrower (more specific) term (generic)

TEP (término específico partitivo) Narrower term (partitive)

TR (término relacionado) Related term that is neither a synonym nor a more generic or specific term.

These relations in the thesaurus can be classified in three types of relations: Equivalency, hierarchy and associative relations. The relations USE, TC and UP fall into the first category, the relations TG through TEP into the second, and the relation TR represents the last category.

Most important for the task of query expansion at hand are synonymy (UP), generalization (TG) and specialization (TE). If a term has a TG relation with second term it means that the second term is a more generic term than the first. An example of this relation is the relation of *travertine* (a kind of stone, in UNL *travertine(icl>rock>thing)* to *rock* (in UNL *rock(icl>natural_object>thing,equ>stone)*).

There are also some additional attributes for every term in the thesaurus like insertion date or author, but they will not be used for the task at hand.

The thesaurus has been developed in Spanish and then translated to UNL. With regard to internationality, this might become a common process, because the creation of a thesaurus manually is very costly, and so the reuse of existent thesauri is a way to significant reduce costs.

In the translation process, the words of the thesaurus are subjected to the same treatment as query words. That means they are normalized and lemmatized in order to be able to look up the word in the dictionary.

The process of translation is interactive, if more than one UNL representation is found the user is asked to select the intended meaning. If the UNL representation for a term is missing, the user is also asked to provide a UNL representation. In both cases, if the user does not provide a translation the term is kept in the thesaurus untranslated so as not to break up structures of hierarchy.

4.2 Expansion of the Query

There are various ways to do query expansion. Examples are spelling correction, morphological expansion, synonyms and expansion with hierarchy.

Because of the translation process, spelling correction is not applicable. Also, as there is a morphological analysis and normalization, it doesn't make sense to enumerate more word forms of the same word, because lemmatization is supposed to let these forms fall together anyway.

The query expansion is therefore done by expanding with synonyms and hierarchy words. These relations are found in the thesaurus. From the hierarchy the direct sons of the word are added, that means more specific words from the level directly below the word in question.

For queries that are not translated, there is no query expansion made. This concerns queries in Spanish. There are two ways to pose a Spanish query. If Spanish is treated as any other language, it is translated and expanded as any other language. If the query is not translated, there is no way of expanding the query as the thesaurus is in UNL.

4.2.1 Expansion with Synonyms

One common way to expand a query is to add synonyms. This is a quite obvious approach, because if somebody looks for documents that contain the word *rock*, he probably would like to get also documents containing the word *stone*, given that it means the same thing.

Query expansion with synonyms is automatic in the translation process. If real synonyms for a word exist, they will have the same Universal Word associated. In the process of translating the Universal Word, every match for this Universal Word in the dictionary is returned. So all real synonyms are included in the query.

For additional expansion with synonyms or quasi-synonyms the relation UP of the thesaurus is used. This relation includes also "quasi-synonyms", that are words that are nearly a synonym. As an example with *rock(icl>natural_object>thing,equ>stone)* (the object *rock* in nature) this gives us *rock(icl>material>thing,equ>stone)* (the material *rock*). Which is not the same thing, but there is only a very small difference.

4.2.2 Expansion with Hierarchy

Besides the expansion by synonyms, there is an expansion done that uses the hierarchy relations found in the thesaurus.

As explained in section 4.1 on page 25, the thesaurus also contains relations of hierarchy. These hierarchy relations are the relation of generalization (term A has a broader meaning than term B, like in *rock* and *travertine*) and the inverse relation of specification (term B (*travertine*) has a more specific meaning than term A (*rock*)). If we look at the thesaurus as a tree, the more specific terms could be regarded as the children of the more general term.

A term of the query is expanded by taking all the children of a term and adding them to the original query. This is done for every term of the query. For example, to the query consisting of the word *rock* the more specific words *travertine* or *limestone* would be added.

This query expansion with children could be done for more than one level or could include also parent nodes. A few experiments have been done with different kinds of expansion (see chapter 7 on page 41).

Chapter 5

The Lucene Search Engine Framework

This chapter is a short introduction to the Lucene framework. More detailed information can be obtained on the Lucene homepage [9] or in the Lucene book [13].

Lucene is an open-source framework for information retrieval. It was started by Doug Cutting in 2000 and is a part of the Apache Jakarta Project since 2002. It is written in Java, but is also available in other languages like C++ or Perl. Lucene itself has only the functionalities needed for indexing and searching, but there are several ”add-ons” for crawling (Nutch[1]) or managing the index (Solr[2]), match highlighting and more.

For an introduction to information retrieval basics see [19]. Here, I will only explain some things in the context of the Lucene search engine.

In order for a search engine to be efficient, the search is not done by directly searching every document that is to be searched. Rather there is an index created where the information can be found more quickly. This index consists of a sorted list of terms. Each term has a postings list associated. The postings list is a list of the documents from the collection where the term occurs. The action of creating the index is called indexing and is the first main task for any search engine.

Secondly, there is, of course, the search, the task of retrieving documents from the index that are relevant to the query. For determining the relevance of a document, a score for every document is calculated and the documents are ranked according to that score.

5.1 Indexing Documents

For indexing, Lucene provides a class `IndexWriter`. This class offers functionality to create and update an index. The index can be optimized.

[1]http://lucene.apache.org/nutch/
[2]http://lucene.apache.org/solr/

The indexer uses an `Analyzer`. The analyzer is in charge of tokenizing the text and doing any normalization wished. There are some analyzers distributed with the library, like, for example, the `WhitespaceAnalyzer`, which simply splits tokens at whitespaces, or the `StopAnalyzer`, which splits tokens at non-letters and also removes stop words. More analyzers are provided by the snowball package of Lucene, which implements porter stemmers for different languages. For the search engine presented here a special analyzer was written that allowed the choice of language at runtime and incorporated the lemmatization rules described in chapter 3 on page 17.

In the indexation process different fields can be stored. The user can decide for each field if it is to be indexed, stored and normalized using the analyzer. Algorithm 5.1 shows some sample code for indexing documents with Lucene.

Algorithm 5.1 Sample Code for Indexing Documents with Lucene

```
// Create an index writer
// The index will be stored at INDEX_LOCATION
// A multilingual analyzer with language INDEX_LANGUAGE is used
IndexWriter writer = new IndexWriter(INDEX_LOCATION,
   new MultilingAnalyzer(INDEX_LANGUAGE), true);
...

// For each file to be indexed
Document doc = new Document();

// Add a field named "path" that contains the path to the file
// The original content is stored along with the document,
// additionally the content of the field is tokenized and indexed
doc.add(new Field("path", file.getPath(), Store.YES, Index.TOKENIZED));

// Add a field named "contents" for the text of the file
// Implies Store.NO and Index.TOKENIZED
doc.add(new Field("contents", new FileReader(f)));
writer.addDocument(doc);
...

// Optimize the index and close it
writer.optimize();
writer.close();
```

Lucene's index algorithm first creates an index for a single document. This is kept in RAM to save system calls and pushed on a stack. A merge factor b is defined. If there are b indexes of the same number of documents on the stack, they are taken from the stack and merged into a single index. This index is pushed onto the stack again. The process with $b = 3$ is shown in figure 5.1 on the facing page

The index is compressed by front coding for the terms in the index and variable byte encoding for the entries of the termlist [5]. Front coding is based on the observation that sorted words often share a common prefix. So to save space, first the common prefix is saved

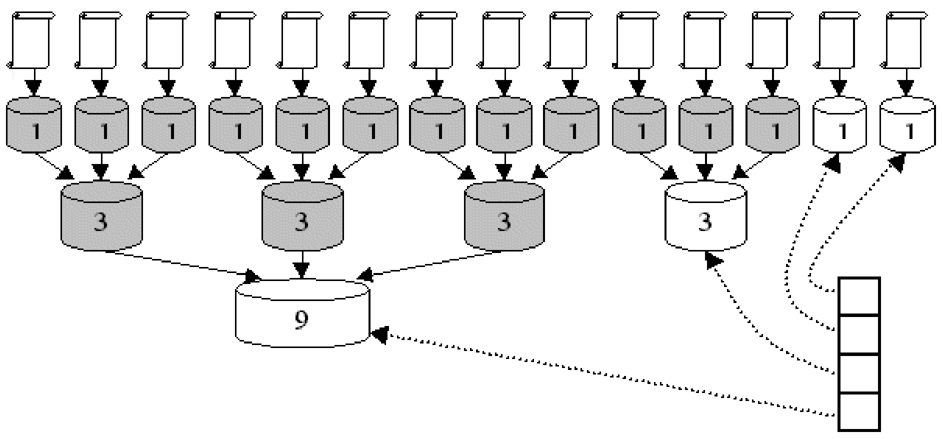


Figure 5.1: Merging of Indexes by Lucene (with $b = 3$)

in the index and then for the words with this prefix only the differing suffixes have to be saved. Variable byte encoding is an efficient way to store the binary representation of an integer in a variable number of bits. The first bit acts as a "continuation" bit and indicates if a new number starts at the current byte or not ([19], chapter 5).

5.2 Retrieving Documents

Lucene supports a number of different query types. The simplest type of query are term queries. In term queries, the index is searched for every term seperately. Multiple terms can be combined with Boolean operators like *AND*, *OR* and *NOT*. OR is the default operator. The search can be performed on different fields. A term can contain wildcards for single or multiple characters. Fuzzy searches can be performed that also find terms similar to the entered term. This can be used for spell correction; *carot* will also return documents that contain *carrot*. A proximity can be given for two search terms, so that in the results they have to appear within the given distance. Phrase queries, where the words have to appear in exactly the given order, are also supported. The weighting of the terms in the search can be adjusted.

All these queries can be entered by hand with the appropiate syntax and fed to the `QueryParser` that Lucene provides (for an overview of the syntax see [10]). The Query-Parser uses an Analyzer; it should be the same as used for indexing.

There is also the possibility to generate queries throught the API. This is done in the search engine presented here, as the input for the search is generated by the program.

Figure 5.2 on the next page shows some sample code for retrieving documents with Lucene.

For computing a ranking of the documents with regard to a query one could compute the similarity of every document with the query. But this is very slow, because many documents

Algorithm 5.2 Sample Code for Searching Documents with Lucene

```
// Open a reader and a searcher
IndexReader reader = IndexReader.open(INDEX_LOCATION);
Searcher searcher = new IndexSearcher(reader);

// Initialize a query parser for field FIELD
// with a multilingual analyzer
QueryParser parser = new QueryParser(FIELD,
       new MultilingAnalyzer(INDEX_LANGUAGE));

// Parse the user input (stored in queryString)
Query myquery = parser.parse(queryString);

// Get matching documents
Hits hits = searcher.search(query);

// Display documents
for (int i=0; i< hits.length(); i++) {
   Document doc = hits.doc(i);
   System.out.println(doc.get("path"));
}
```

will not have any terms in common with the query. It is better to use the index and to compute similarties only for the documents in the postings lists of the query terms.

Lucene's search algorithm for disjunctive searches is described in detail in [6]. For a query consisting of various terms the postings lists of all terms are gathered. In each postings list the documents are sorted by document ID. The entries in the postings lists for the document with the lowest document ID are now taken out and the score for this document is computed. It is saved in a ranked list of the k documents with the highest scores. This process continues computing the score for every document until there are no postings left in the postings lists.

With this algorithm only the postings lists and the scores for k documents have to be stored. Every document is scored, so the k documents returned are really the k documents with the highest scores. The algorithm scales well with large queries.

For conjunctive queries the algorithm is altered in a way that for the postings list of all terms the first document ID is compared and a hit is only recorded if a document appears in all of the queues.

5.3 Scoring Documents

Lucene uses the vector space model for scoring documents against queries. In the vector space model, every document is stored as a vector where every dimension represents one term of the vocabulary of the collection. The terms that occur in the document have a non-zero value that represents the weight for that term in that document. Queries are represented by vectors in the same way. Note that this model does not retain word order ("bag of words") ([19], chapter 6).

When comparing a document with a query, we actually compare the vectors of the documents and the query in vector space. The similarity is a measure of how near they are to each other in vector space.

The formula of Lucene for computing the score of a document *d* with regard to a query *q* consisting of terms *t* is ([11]):

$$score(q,d) = coord(q,d) \cdot queryNorm(q) \cdot \sum_{t \in q} (tf(t \in d) \cdot idf(t)^2 \cdot boost(t) \cdot norm(t,d))$$

where

coord(q,d) is a score factor based on how many of the query terms are found in the specified document.

queryNorm(q) is a normalizing factor used to make scores between different queries comparable. It doesn't influence ranking.

tf(t in d) is the term frequency, defined as the square root of the number of times term t appears in the currently scored document d.

idf(t) is the inverse document frequency, that is the inverse of the number of documents where the term appears.

boost(t) is the weight of the term t in the query.

norm(t,d) is a collection of factors of the document like length, boost of the document and the fields.

The tf factor gives weight to a document where the search term appears. The more times the term appears, the better. Likewise, the idf factor gives less weight to a term if it appears in a lot of documents and more weight to terms that are rare. This follows the reasoning that a term that appears in every document of the collection is not useful for distinguishing relevant documents from not relevant documents. But a search term that is seldom used in the collection will give a high significance to the document we found it in. The combination of tf and idf gives a high value when the term we are searching occurs often in the current document and in general in few documents. The usage of a normalization factor of documents is a common way to avoid long documents unproportionally influencing the search results.

5.4 Why Lucene?

The focus of the presented work was to create a prototype of a search engine based on Universal Words. That means, the most important part is the pre-processing before the actual search and not the implementation of specific information retrieval techniques.

It therefore makes sense to reduce the amount of implementation time for the basic information retrieval tasks as much as possible. Lucene presents a framework that provides these tasks in a very performant way. The decision to use Java Lucene was also made to be as platform independent as possible.

Lucene's algorithms for searching, indexing and scoring are independent from the language of the documents or the queries. The only place where language dependence comes in is in the choice of the analyzer. Lucene is using Java UTF-8 to represent Strings internally (see 6.1 on the facing page), so there are no problems in using accented characters or different alphabets. This makes Lucene well-suited for a multilingual search engine.

Lucene is used in projects, like, for example, Eclipse[3], Recht für Deutschland[4], IBM OmniFind Personal e-mail Search[5] and Monster[6] (according to the Lucene homepage, [9]).

[3]`http://www.eclipse.org/`
[4]`http://www.recht.makrolog.de`
[5]`http://www.alphaworks.ibm.com/tech/emailsearch`
[6]`http://jobsearch.monster.com/`

Chapter 6

Design and Implementation

In this chapter, some of the implementational issues will be adressed. In section 6.1, Unicode and encodings for Unicode are briefly explained. In section 6.2 on the following page the requirements for the multilingual search engine are listed. From these requirements the conceptual design in section 6.3 on page 38 is derived to get an overview about the structure. In section 6.3 on page 39, the design of the database for the storing of the dictionary and the thesaurus is given.

6.1 Unicode and Encodings for Unicode

Anybody who has ever tried to write something in an "exotic" language on the computer knows about how things can go really wrong. This goes from reopening your document and finding some strange rectangles in place of some special characters of your text to total unreadable chaos of unrelated letters or only question marks.

As our world grows more and more international with the internet, a lot of these questions have been adressed in the last few years.

In the multilingual search engine that is discussed here, there are three different writing systems to consider. Latin for English and Spanish – with Spanish contributing special characters (ñ and accented letters like á) – Cyrillic for Russian, and Arabic letters for Arabic. The challenge is to integrate all these different scripts in one application.

Since the year 1987, there have been attempts to create one character set for every existing writing system. In the end, the Unicode consortium decided not to define ways to store characters on disk, but to create a theoretical representation of each character. The first Unicode standard was published in 1991.

Unicode maps each letter to a code point. It is written as a hexadecimal number with a U to indicate Unicode. The English letter A for example is U+0041. The shape of a code point is left to the software, it is not encoded in Unicode. Also, the exact way to store a character is not part of the Unicode standard.

Unicode	Byte1	Byte2	Byte3	Byte4
U+000000-U+00007F	0xxxxxxx			
U+000080-U+0007FF	110xxxxx	10xxxxxx		
U+000800-U+00FFFF	1110xxxx	10xxxxxx	10xxxxxx	
U+010000-U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

Table 6.1: UTF-8 Encoding

At the moment (September 2008) more than 100.000 characters have been assigned a Unicode representation [4].

There are various ways to encode Unicode. The first way to store Unicode was to encode every Unicode code point in two bytes. So the letter A at point U+0041 is encoded as `00 41`[1]. This is called UCS-2. But Unicode characters that need more space than 2 bytes cannot be stored in UCS-2. UTF-16 is a newer variant of UCS-2 that uses two byte to encode the Unicode code point. If the characters need more space than two byte, four byte are used.

The most common encoding for Unicode is UTF-8. UTF-8 is a variable length encoding, it stores Unicode code points in one or up to six bits. UTF-8 can store any Unicode code point. Every code point from 0-127 is stored in one byte. This means that text using only English characters looks the same in ASCII and UTF-8. If one byte of a letter is lost, it is possible to resynchronize at the next letter, and so confine the damage to the letter where the loss occurred. It will not affect the rest of the text (as it will, for example, in UTF-16). It is also very improbable that a byte sequence seems valid UTF-8 but is not [23].

Today more and more software uses UTF-8, but there is still a lot of software that doesn't. But it is still sometimes diffucult to get things to work together. Java uses only UTF-8 internally. My multilingual search engine relies on all files, inputs and databases being in UTF-8.

6.2 Requirements

There are three different types of people who use the system. Obviously, there are the people who execute queries and want to obtain documents as an answer. I will call them "users" in the following document. Secondly, there is somebody who maintains the index of documents, called "admin" in the following. And lastly there are the linguistic resources that also have to be maintained by somebody. I will call this person "linguist".

Functional requirements are tasks the system has to execute, they are denoted as "RF". User interface requirements are denoted as "RU", operational requirements (that are for example formats of files, ...) as "RO", resources as "RR" (following the VAI standard of documentation [2]).

[1]On some machines it could be encoded as `41 00`, therefore it is recommended to add a Byte Order Mark (BOM) to the beginning of each String. This BOM is the Unicode code point U+FEFF, which apart from representing the BOM represents a null-width non-breaking space.

Requirements are marked as ”essential” if they are a crucial part of the system, ”desirable” if they would add a considerable enhancement to the system and are considered important and ”optional” if their realization would add functionality to the system that is merely nice to have.

Functional Requirements

RF1 The user can search for single word or multi word terms (terms consisting of at most three words) in all of the supported languages (essential)

RF2 The user can execute phrase queries (groups of words that are to be found together but are not one term) in Spanish (desirable)

RF3 The admin can add documents to the index (essential)

RF4 The admin can delete documents from the index (essential)

RF5 The linguist has the possibility to load translations of UNL words into a language into the system (essential)

RF6 The linguist has the possibility to load a UNL thesaurus into the system (essential)

I will not include phrase search for all languages, because the only resource available for translation, the UNL dictionary, will not contain verbs or adjectives. Without the possibility to translate verbs, adjectives or prepositions, a translation of a phrase is impossible, and so I won't be able to search for them in the Spanish index.

I will not implement the functionality that a linguist can add a new language to the system. That is because to add a new language, a Lucene analyzer of the language would have to be provided. In my opinion, this is considered to be work of a developer who wants to expand the system and should be done on code-level.

Likewise I will also not provide functions for maintaining the dictionary or the thesaurus, because other tools are being developed for that. An exception may be the possibility to load partial dictionaries, that is to change the translation of a few words without completely reloading the dictionary.

User Requirements

RU1 For each returned document there will also be returned a short description of the document that will contain a window of the text with the (Spanish) matched words highlighted (desirable)

RU2 The user must choose the language of the query in the web-interface (essential)

RU3 The user interface is at least in Spanish, regardless of chosen query language (essential)

Performance Requirements

RP1 Multiple users can search the index at the same time (essential)

Operational Requirements

RO1 Queries are entered in UTF-8 encoding (essential)

RO2 The system can handle left-to-right languages (like English) and right-to-left languages (like Arabic) (essential)

RO3 The thesaurus has to be provided in XML-Format (essential)

6.3 Conceptual Design

In the following, the conceptual design of the multilingual search engine will be described.

The supported languages of the multilingual search engine are Spanish, English, Russian and Arabic. For every language there is some knowledge required for the morphologic analysis. Also a dictionary of the language words and UNL has to be provided for the translation process. For the query expansion a thesaurus is necessary. An index of the Spanish documents has to be created beforehand to be used in the search.

The search engine is designed to work on a collection about Spanish cultural heritage provided by the Ministry of Culture. Therefore, the thesaurus that will be used in the expansion of the query is dependent on that domain. The vocabulary of the dictionary is also taken from the domain. But as the knowledge is separated from the processing, to change the domain, it is enough to exchange the dictionary and the thesaurus.

In the following, the process that one user input goes through the system is described. As an intuitive example, the result in every step of the user query *Minerals* with language *English* is given. It is assumed that the user interaction takes place via a web browser. The different steps of processing are visualized in 6.1 on page 40.

The expected **input** to the system the user has to make is a query in one of the supported languages. The query may consist of one or various words. The words may be in upper or lowercase. Punctuation or asterisks may be entered, but will be ignored in processing. Operators such as *and* or *or* will be treated like normal words. The user also has to specify the language of the input. The input and its language are passed on as an output of the module. As an example, an input could be *Minerals* and the language *English*.

The **morphological analysis** takes the user input as its entry. During the analysis, the text is split into single words (tokens). Stop words are removed from the input. The remaining tokens are normalized. Text normalization differs for every language. It can involve removing of accents (Spanish) or vowels (Arabic). For more details, see chapter 3 on page 17. The normalized text is analyzed morphologically. Every noun in the query is replaced by its lemma. In the example, *Minerals* will be replaced by its lemma, that is, the singular *mineral*.

The query consisting now of a series of lemmata is now passed on to the **translation to UNL**. With the help of the dictionary, every word in the query is translated from the source language to UNL. Words that are not in the dictionary are removed. There might be various translations for a word if the word is ambiguous. In this case, all words will be put into the query, there is no ambiguity resolution. In the example, the translation of *mineral* will give the UNL representation *mineral(icl>material>thing)*.

This UNL query is now passed on to the **query expansion**. There the query is expanded with the help of the thesaurus. In the example, a few words would be added to the word *mineral* like *malachite* or *gold* – all in UNL. The final query is now *mineral(icl>material>thing) malachite(icl>mineral>thing) gold(icl>noble_metal>thing)*.

The last step in the pre-processing of the query is the **translation to Spanish** of the expanded query. This is done again with the help of the dictionary. In the example this would give us *mineral malaquita oro*.

Then the final query is passed on to the **search engine** and a list of links to all retrieved documents is returned. This list is presented to the user in an interface where he can easily navegate to every document he wants to. In the example, the list of documents would hopefully include a lot of useful information about minerals in the Spanish cultural heritage.

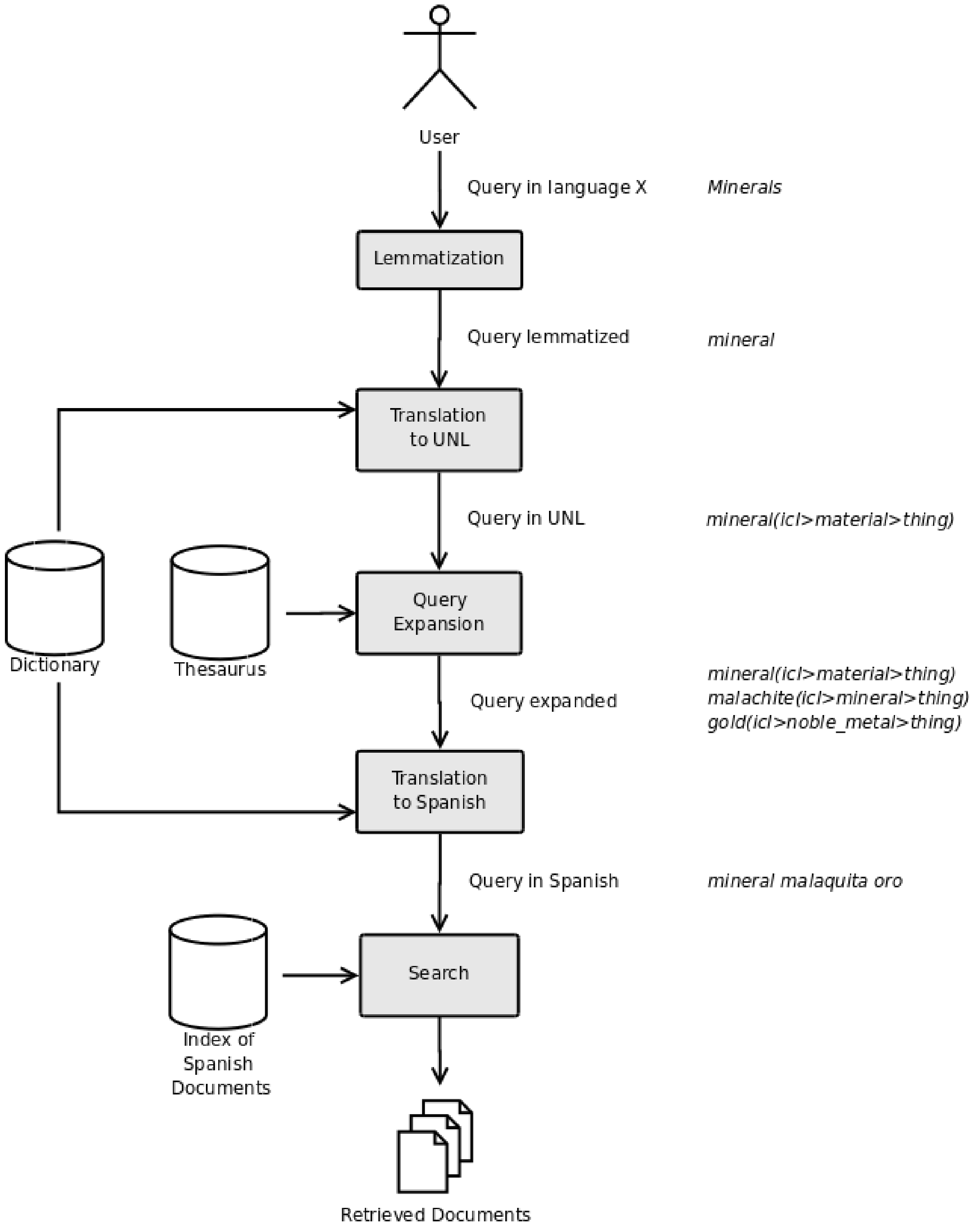


Figure 6.1: Conceptual Design

Chapter 7

Evaluation and Validation

An exhaustive evaluation of the search engine that would deliver statistically relevant data was not done. This was not possible because the document collection is very small (153 documents), and the linguistic resources used – the thesaurus and the dictionaries – were developed in parallel, and at the time of writing were only partially finished[1].

This means, that the results obtained in testing are merely a check if the approach of basing a search engine on UNL as an interlingua can work.

7.1 Fulfillment of Requirements

All requirements have been fulfilled, the functionality is complete.

Requirement RU1 "For each returned document there will be returned also a short description of the document that will contain a window of the text with the (Spanish) matched words highlighted (desirable)" is fulfilled. The system is returning windows with matching words, but is not yet able to correctly highlight phrases.

7.2 Quality Improvement with Query Expansion

The testing of the effectivity of the query expansion is difficult to measure at the current state of the linguistic resources.

The document collection contains 153 documents. Of these, 142 are plain text documents, five are pdf documents, four are Word documents and two are zip archives. Only the text documents have been considered.

The evaluation of the effectiveness of the query expansion was done following the methodology described by Hersh, Price and Donohoe in [14]. First, a few queries were run on

[1]The thesaurus contained about 500 words, the Spanish and English dictionaries 1200 words, the Russian dictionary 1000 and the Arabic dictionary 380 words.

	Precision	Recall	F-Measure	Improved queries
Without expansion	0.34	0.56	0.42	-
Synonym expansion	0.43	0.66	0.52	0.13
Child expansion - one level	0.60	0.94	0.73	0.50
Child expansion - two level	0.60	0.94	0.73	0.50
Child expansion - all levels	0.60	0.94	0.73	0.50
Parent expansion - one level	0.42	0.63	0.51	0.13
Parent expansion - two level	0.39	0.66	0.49	0.13
Parent expansion - all levels	0.39	0.66	0.49	0.13
Synonyms and direct sons	0.58	0.94	0.71	0.5

Table 7.1: Evaluation Results of Query Expansion

the system without any expansion on the query. Of course, through the translation, there might already be synonyms added to the query. The same set of queries was then run with expansion by synonyms. Then once again the same query set was expanded by children, first on one level, then two and finally all levels down to the leaves. The same is done with the parent elements.

The set of queries consisted of eight queries from one to three words. A huge problem for evaluation was, that at the time of testing only one third of the thesaurus was available and unfortunately, many words of this third had either no translation or were not present in the collection (or only once). Also, the sets of relevant documents for a query where assigned by me and not a domain expert, which might lead to errors.

The queries were entered in English, because the dictionary of English contains a lot more words than in the other two languages. Apart from the first translation step, the performance of the search engine is independent from the query language.

For every kind of expansion, the average precision, recall and f-measure were computed. Precision is the proportion of returned documents that are relevant. Recall is the proportion of relevant documents that have been found. The f-measure is a combination of both, it is calculated by

$$F = \frac{2PR}{P + R}$$

where P denotes precision and R recall.

Precision and recall are calculated over all the retrieved documents, because there are often very few documents retrieved (two or three). So to compute precision or recall at ten does not make sense. The averaged results of the test runs are shown in table 7.1. Improved queries is with regard to the results of not expanding the query at all.

The highest precision (0.60) is obtained with expanding down. The number of levels does not influence much. This is because the thesaurus has only four levels, so if we have a term from the third level, there is only one level to expand.

Highest recall (0.94) is also obtained with expanding down, therefore f-measure is also highest there.

These results are not statistically significant in any way, but they show what is intuitively clear. Expanding the query with a more general term does not help much. It may even be harmful to precision, because there are a lot more documents returned but with the possibility that none of them are relevant (if somebody wants documents about *marble* he is not interested in documents that speak about *stone* in general).

On the other hand, adding more specific terms to the query helps to find more documents that speak about the things the user wanted to know. Also, it might help to show the user what terms were used for search, so that he can refine his query.

The expansion with synonyms does not bring a big improvement. This is a bit surprising, because one should think that this is exactly what we want to overcome: vocabulary mismatch between query and documents. This result is probably due to the fact that only in three cases (out of eight queries) there were synonyms added and in only one case this consisted in really new words (*conservation* received *safeguarding*). The result of expansion with synonyms would improve with more synonyms in the thesaurus.

All above-mentioned expansion methods are implemented in the multilingual search engine. The default way to expand is a combination of expanding with synonyms and expanding with one level of children. Only one level is used, because the addition of more levels does not improve the results, but rather consumes time.

Chapter 8

Conclusion

This work presents a fully functional prototype of a multilingual search engine using UNL as a common interlingua. Search queries using the currently supported languages English, Arabic and Russian return a list of Spanish documents containing the translated search terms. The search is further improved by thesaurus based query expansion. An advanced query interface for Spanish was added to make the full power of the Lucene framework available for phrase searches and wild card searches.

The Lucene Framework has proven to be a very good choice and allowed me to concentrate my efforts on the new aspect of multilingual search and keep the focus on the integration of UNL and the query expansion.

I conducted a preliminary evaluation showing mainly the correct functioning of the prototype for all languages. The results indicate that the thesaurus based query expansion can improve the search result quality. However, the very limited document base, as well as the early state of the linguistic resources made a real evaluation impossible at the current point in time.

Besides the obvious need for a more exhaustive dictionary and thesaurus, there are several other areas where the search engine can be improved. At the present state the lemmatization is very rudimentary. It could be improved a lot by adding more rules for the different languages. Additionally, there are currently only rules for nouns, which is only a small part of all words. To cover all words, a tagger would be necessary to select the correct rules for every kind of word.

Another interesting discussion point is whether the last step – the translation to the index language – could be saved if we index the documents in UNL. This would also eliminate any ambiguity introduced by the translation to the index language. Of course, to be able to do this, the translation to UNL has to be of good quality, and, above all, has to enable us to disambiguate. But as we don't have to translate sentence structures, because there are only words indexed, the task should not be impossible. Still, it would also be necessary to evaluate if the work involved in translating all documents of a potentially huge collection is worth the gain in retrieval quality.

At the moment, the user has to choose the language. One could add automatic language recognition to the system. This would be easier with a whole sentence as user input and not only a few keywords.

Acknowledgements

First of all I need to thank Holger Rode for a lot of help and encouragement. I thank Jesús Cardeñosa for welcoming me into the VAI group and giving such an interesting project to an unknown foreigner. I also need to thank the members of the Patrilex project who provided me with help on the different areas: Adriana Toni (Arabic lemmatizer and dictionary), Ángeles Maldonado (the thesaurus), Carolina Gallardo (Spanish lemmatizer, query expansion and Lucene) and Igor Boguslavsky (Russian lemmatizer and dictionary). And finally thanks go to all those who read this work and helped me with suggestions: Arielle Weisman, Christine Keller, Henning Rode and Johannes Imo.

Appendix A

User Guide

How Does the Search Work

When you enter some search words there are several steps to be done before you receive your results.

1. Normalization
   When searching for *Minerals* you will also want to find documents that contain the word *minerals*. Because of that, words are normalized to a standard (here lowercase). Also you would probably like to find documents that contain *mineral*, so every plural noun in the search words is replaced by its singular form.
2. Translation to UNL
   The search words are translated to an artificial language called UNL. From this point on, the initial query language is not important anymore. The word *mineral* is *mineral(icl>material>thing)* in UNL.
3. Query expansion
   Synonyms and more specific words are added to the original search words. For example, to the word *mineral* the words *malachite* or *gold* would be added (in UNL).
4. Translation to Spanish
   Finally the words are translated to Spanish, what would give us *mineral malaquita oro* in the example.
5. Search for documents
   The search will return all documents that contain at least one word of the Spanish search words.

Search Modes

There are four different modes for the multilingual search engine available. They differ in which of the the above mentioned steps they perform.

Normal search This is the default mode. The search includes the steps normalization, translation to UNL, query expansion, translation to Spanish and search.

No query expansion The step query expansion is excluded from the procedure, so the search consists only of normalization, translation to UNL, translation to Spanish and search.

No translation, no expansion In this mode the search words are only normalized. As there is no translation, this mode is useful in searches for names or Spanish words.

Advanced search When this mode is selected, more specific searches can be conducted. For an explanation of the possibilities see the explanation of the advanced search below.

Selection of Language

The selection of the correct language is very important for the success of the search. It influences the dictionary that is used and the kind of normalization that is done. The Spanish word *árboles*, for example, will be correctly normalized to the singular *arbol* when Spanish is selected. But if English is selected it becomes *arbole*. This word will not be in the English dictionary and if the word is searched without translation it will not find documents that contain *árbol* or *árboles*, because it is not the same word.

Advanced Search

There are many different ways to search for something if you use the advanced search. Note that these features are only available if you search in Spanish or search for names. If your search words are translated all of the special symbols will be ignored.

Phrase searches A phrase is a group of words that you want to find together. It has to be entered surrounded by quotes. A search for *"monasterio real"* will only find documents where these words appear next to each other.

Wildcard searches Wildcard searches allow you to search for words leaving undefined one or more letters. The symbol *?* can replace one letter, the symbol * replaces various letters. A search for *te?t* will find documents for *test*, *text* and so on. Note that you cannot use * or *?* as the first character of a search or in phrase searches.

Fuzzy searches Fuzzy searches find also words that are similar in spelling to the original search word. The ~ symbol after the search word is used for this type of search. A search for *madera~* will also find documents for *manera* or *madero*.

Proximity searches This type of search is similar to a phrase search, but here you can specify the maximum distance where two words must appear. A search for *"monasterio visita"~10* will find every document where the words *monasterio* and *visita* occur less than 10 words from each other.

Fields A document contains different fields. The path of a document is contained in the field ”path”, the title in ”title” and the contents of the file in ”contents”. So if you want to find documents that have the word *monasterio* in the title try *title:monasterio*

Boosting of terms You can give more weight to a word or a phrase you are searching by adding the symbol ˆ followed by a number, as, for example, in *monasterioˆ3 real*. The higher the number is, the more important the word is for your search.

Boolean operators By default documents that are returned as a result of your search must contain at least one of the search words. That is equivalent to writing *OR*. If you want only documents that contain all of the search terms you can join them by *AND* as in *monasterio AND visita*. The + symbol denotes a term that must be present in all returned documents, as the - symbol forbids a term to be present in one of the returned documents. So a search for *+monasterio visita -escorial* will give you all documents that contain *monasterio*, may contain *visita* and do not contain *escorial*. *AND* and *OR* must be in upper case. + and - must be directly in front of the word, without a space.

Appendix B

Installation Guide

The Patrilex multilingual search engine is implemented as a Java Servlet and runs with Apache Tomcat. Additionally there is a MySQL database required. If you are familiar with the installation and configuration of these applications you may skip these parts of this document. Please note that your MySQL has to be set to UTF-8.

The installation has been tested with Ubuntu and Debian using MySQL version 5.0, Apache HTTP Server version 2.2.8 and Apache Tomcat version 5.5. There is no guarantee it will work with other combinations.

Installation of MySQL

These instructions are based on the version 5.0 of MySQL installed on a Debian or Ubuntu Linux. If you are using a different version or operating system you will have to adjust them.

1. Install MySQL following the instructions on `http://www.mysql.com`
2. Change the character encoding of the server to UTF-8. This is done by restarting mysqld with the option `--character-set-server=utf8`

   ```
   mysqladmin -u root -p shutdown
   mysqld --character-set-server=utf8
   ```

   To make this change permanent the startup script for MySQL has to be modified. On Debian-based systems this means modifying the file `/etc/init.d/mysql` . Add the above option to the startup of mysqld (should be in `start`). That means the line

   ```
   /usr/bin/mysqld_safe > /dev/null 2>&1 &
   ```

   changes to

   ```
   /usr/bin/mysqld_safe --character-set-server=utf8 > /dev/null 2>&1 &
   ```
3. Log into MySQL and create a database that uses UTF-8 to use for the Patrilex multilingual search engine. Here the database will be called `patrilex`.

```
create database patrilex default character set utf8;
```

Create a user for the Patrilex search engine and grant him rights on the patrilex database. As an example we will call the user `patrilexadmin` and set his password to `mypassword`.

```
grant all privileges on patrilex.* to patrilexadmin@localhost
  identified by "mypassword";
flush privileges;
```

4. The language part of the environment variables if you log into MySQL and connect to the newly created database should now look like this (execute the command `show variables` to see this table):

```
| character_set_client            | utf8                        |
| character_set_connection        | utf8                        |
| character_set_database          | utf8                        |
| character_set_filesystem        | binary                      |
| character_set_results           | utf8                        |
| character_set_server            | utf8                        |
| character_set_system            | utf8                        |
| character_sets_dir              | /usr/share/mysql/charsets/  |
| collation_connection            | utf8_general_ci             |
| collation_database              | utf8_general_ci             |
| collation_server                | utf8_general_ci             |
```

Installation and Configuration of Apache HTTP Server

These instructions are based on the version 2.2.8 of the Apache HTTP Server installed on a Debian or Ubuntu Linux. If you are using a different version or operating system you will have to adjust them.

1. Install Apache HTTP Server following the instructions on `http://httpd.apache.org/`
2. Edit `apache2.conf` and `sites-available/default` in the installation directory (on Debian `/etc/apache2/`) according to the instructions from `http://httpd.apache.org/` so that you are able to host a webpage.
3. Restart Apache for the changes to take effect

Installation and Configuration of Tomcat

These instructions are based on the version 5.5 of the Apache Tomcat installed on a Debian or Ubuntu Linux. If you are using a different version or operating system you will have to adjust them.

1. Install Apache Tomcat following the instructions on `http://tomcat.apache.org/`
2. Adjust the settings in the file `/etc/default/tomcat5.5` to your system. Usually this should be it:

   ```
   JAVA_HOME=/usr/lib/jvm/
   CATALINA_BASE=/var/lib/tomcat5.5
   ```
3. Permit the Patrilex multilingual search engine the access to the database by adding the following to the file `/var/lib/tomcat5.5/conf/context.xml` . You can put it in the default context, the context of / or in the context of the patrilexweb servlet (see `http://tomcat.apache.org/tomcat-5.5-doc/jndi-datasource-examples-howto.html`). Here it is supposed that the database is called `patrilex` and there is a user `patrilexadmin` with password `mypassword` existing.

   ```
       <ResourceParams name="jdbc/patrilexdb">
           <parameter>
             <name> driverClassName </name>
             <value> com.mysql.jdbc.Driver </value>
           </parameter>
           <parameter>
             <name> url </name>
             <value> jdbc:mysql://localhost/patrilex </value>
           </parameter>
           <parameter>
             <name> username </name>
             <value> patrilexadmin </value>
           </parameter>
           <parameter>
             <name> password </name>
             <value> mypassword </value>
           </parameter>
     </ResourceParams>
   ```
4. Move jdbc Driver (should be in `/usr/share/java/mysql.jar`) to the Tomcat library folder (should be in `/usr/share/tomcat5.5/common/lib`).

Installation of Patrilex Multilingual Search Engine

1. Stop Tomcat
2. Copy the file `patrilexweb.war` that contains the Patrilex multilingual search engine to the Tomcat web application folder (should be `/usr/share/tomcat5.5/webapps`)
3. Start Tomcat again
4. Open a browser. You should be able to see the the welcome page of the Patrilex multilingual search engine at `http://localhost:8180/patrilexweb/welcome`[1]

[1] 8180 is the default port for Tomcat

Configuration and Initialization of Patrilex Multilingual Search Engine

Before you can use the Patrilex multilingual search engine, the search engine must be told where to find the database and some other resources.

1. Stop Tomcat
2. Edit the configuration file `init.conf` in the web application folder (should be `/usr/share/tomcat5.5/webapps/patrilexweb`) giving the paths to all the dictionary files and thesaurus files you want to use. Note that all thesaurus files you want to import have to be in UNL. You can translate a thesaurus later with the Patrilex management tool.
3. Run the patrilex management tool by running the script `patrilexmanager.sh` from the web application folder in your shell
4. Select point 1 (”Initialize Patrilex”) from the menu. This will import the dictionaries and the thesaurus from the files you supplied in the cofiguration file and create an index.
5. Try out some searches at `http://localhost:8180/patrilexweb/search` .
6. You can use the patrilex management tool also to later manage the resources, for example, to translate a thesaurus or to add more translations to the dictionary.

Bibliography

[1] Igor Boguslavsky. Some controversial issues of unl: Linguistic aspects. *Research on Computing Science, Vol. 12, Universal Networking Language Advances in Theory and Applications*, 2005. (Cited on page 14)

[2] Jesús Cardeñosa. Documentación asociada a cualquier desarrollo del grupo vai, 2006. (Cited on page 36)

[3] Jesús Cardeñosa, Carolina Gallardo, and Edmundo Tovar. Standardization of the generation process in a multilingual environment. *Research on Computing Science, Vol. 12, Universal Networking Language Advances in Theory and Applications*, 2005. (Cited on pages 13 und 14)

[4] The Unicode Consortium. *The Unicode Standard, Version 5.1.0.* Addison-Wesley, Boston, MA, 2007. http://www.unicode.org/versions/Unicode5.1.0/. (Cited on page 36)

[5] Doug Cutting. Lucene lecture at pisa, 2004. http://lucene.sourceforge.net/talks/pisa/. (Cited on page 30)

[6] Douglass R. Cutting and Jan O. Pedersen. Space optimizations for total ranking. *Proceedings of RAIO 97*, 1997. (Cited on page 32)

[7] Asociación Española de Normalización y Certificación (AENOR). Norma une 50-106-90. directrices para el establecimiento y desarrollo de tesauros monolingües, 1990. (Cited on pages 25 und 26)

[8] Real Academia Española. Diccionario panhispánico de dudas, 2008. http://buscon.rae.es/dpdI/. (Cited on page 20)

[9] The Apache Software Foundation. Lucene homepage. http://lucene.apache.org (last visit on September 7, 2008). (Cited on pages 29 und 34)

[10] The Apache Software Foundation. Apache lucene - query parser syntax, 2008. http://lucene.apache.org/java/2_3_1/queryparsersyntax.html. (Cited on page 31)

[11] The Apache Software Foundation. Apache lucene - similarity, 2008. http://lucene.apache.org/java/2_3_2/api/core/org/apache/lucene/search/Similarity.html. (Cited on page 33)

[12] Carolina Gallardo. Prologue. *Research on Computing Science, Vol. 12, Universal Networking Language Advances in Theory and Applications*, 2005. (Cited on pages 11 und 14)

[13] Erik Hatcher and Otis Gospodnetić. *Lucene in Action*. Manning Publications, 2004. (Cited on page 29)

[14] William Hersh, Susan Price, and Larry Donohoe. Assessing thesaurus-based query expansion using the umls metathesaurus. *Proceedings of the 2000 Annual AMIA Fall Symposium*, 2000. (Cited on page 41)

[15] Robert Hoare. Morewords.com. http://www.morewords.com. (Cited on page 19)

[16] J. Akshay Iyer and Pushpak Bhattacharyya. Using semantic information to improve case retrieval in case-based reasoning systems. *Research on Computing Science, Vol. 12, Universal Networking Language Advances in Theory and Applications*, 2005. (Cited on page 13)

[17] Jesús Cardeñosa Lera. Introduccion al lenguaje unl. *Proyecto MINORLENG*, 2005. (Cited on pages 11 und 12)

[18] Jesús Cardeñosa Lera. Memoria científico-técnica del proyecto. *Convocatoria de ayudas de Proyectos de Investigación*, 2005. (Cited on page 14)

[19] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. http://www-csli.stanford.edu/ hinrich/information-retrieval-book.html. (Cited on pages 17, 18, 29, 31 und 33)

[20] Ronaldo Teixeira Martins and Maria das Graças Volpe Nunes. On the aboutness of unl. *Research on Computing Science, Vol. 12, Universal Networking Language Advances in Theory and Applications*, 2005. (Cited on pages 11 und 14)

[21] UNL Center of UNDL Foundation. Universal networking language (unl) specifications version 2005, 2005. http://www.undl.org/unlsys/unl/unl2005. (Cited on page 11)

[22] Xiaodong Shi and Yidong Chen. A unl deconverter for chinese. *Research on Computing Science, Vol. 12, Universal Networking Language Advances in Theory and Applications*, 2005. (Cited on page 13)

[23] Joel Spolsky. The absolute minimum every software developer absolutely, positively must know about unicode and character sets (no excuses!), 2003. http://www.joelonsoftware.com/articles/Unicode.html. (Cited on page 36)

[24] Wikipedia. Machine translation. http://en.wikipedia.org/wiki/Machine_translation. (Cited on page 10)

[25] Wikipedia. Query expansion. http://en.wikipedia.org/wiki/Query_expansion. (Cited on page 25)

[26] Étienne Blanc. About and around the french enconverter and the french deconverter. *Research on Computing Science, Vol. 12, Universal Networking Language Advances in Theory and Applications*, 2005. (Cited on page 13)

All links were last followed on December 10, 2008.

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

(Stefanie Wiltrud Kessler)