Metalanguage Enrichment & Concept Lexicalization
A EuroWrdNet Implementation

Majdi Mohammed
B.A. (Mod.) Computer Science, Linguistics and a Language
Final Year Project May 2007
Supervisor: Dr. Martin Emms

School of Computer Science and Statistics
O'Reilly Institute, Trinity College, Dublin 2, Ireland
Declaration

I hereby declare that this thesis is entirely my own work and that it has not been submitted as an exercise for a degree at any other university. I authorise its use in Trinity College Library.

__________________________________________ May 4th, 2007
Majdi Mohammed
Acknowledgements

My sincerest thanks to my family for their constant support and to my girlfriend Megan and her family for all the support that I have received from them. I would also like to thank my classmates for being helpful and fun, and Carl Vogel for being an admirable teacher and friend. And last but by no means least, I would like to show my utmost appreciation to the supervisor of this project, Dr. Martin Emms, for his guidance and for always being patient with me.
Abstract

The aim of this project is to generate paraphrases and metaphors for English words using German composite words.
Table of Contents

Chapter 1 7.

1.1 Introduction
1.2 Utilizing this feature for computational linguistic purposes
1.3 Developing this concept with the German Language
1.4 Why seek circumlocution

Chapter 2 11.

2.1 Introduction
2.2 What is EuroWordNet?
2.3 Type of relations in EuroWordNet
2.4 Synonymy
2.5 EuroWordNet Import/Export format
2.6 Inter-lingual-index
2.7 Conclusion

Chapter 3 18.

3.1 Introduction
3.2 Description of the Software
3.3 The Logic behind the Design
3.4 Algorithm
3.5 Classes’ Specification
3.6 The logic of the program

Chapter 4 36.

4.1 Limitations
4.2 Conclusion

References 37.
List of Tables

Table 1: Language of Internal Relations between synsets in EuroWordNet 13.
Table 2: Language-Internal Relations between other data types in EureoWordNet 14.
Table 3: Software provided by the supervisor 19.
Table 4: Database files and their inverted indices 30.

List of Figures

Figure 1: Sequence of steps to perform the translation 9.
Figure 2: Synsets related to “car” in its first sense in WordNet1.5 12.
Figure 3: Sequence diagram 1 34.
Figure 4: Sequence diagram 2 35.
Chapter 1

1.1 Introduction

The project was inspired by a paper written by Tony Veale and Shanshan Chen\(^1\), entitled, *Unlocking the Latent Creativity of Orthographic Structure*. In this paper, Veale and Chen sought to delve into the depths of the English language through an investigation of the Chinese language using both WordNet and HowNet databases. The Chinese written language uses sequences of logograms (more commonly referred to as characters) to create words. Logograms do not have the same phonetic function as letters do in an alphabet; nevertheless what concerns us here is that they often have an associated meaning. However, one cannot simply relate English words to their associated meanings in the same way that one would relate Chinese logograms to their associated meanings. The reason for this is that the meaning in Chinese can be a combination of several meanings, depending on how many logograms the sequence contains. Hence, sometimes a logogram can have its meaning independently of other logograms, but could then join together with these logograms to form a single word, and thus another meaning. Of most importance here, however, is that the relationship between the meanings of the sub-parts and the meanings of the whole is not random. This feature of the Chinese language can provide an etymological hypothesis of the origin of a word which would lead us to believe that it was coined as a concatenation of independent words.

In some cases the reasoning behind composing such words is rather straightforward and obvious, as in many noun-noun composite words in English. In other cases, the grounds for such constructions can be perceived as metaphorical. Often this more symbolic sense of the word is due to the passing of time, meaning that at one stage the coined words had a much more literal meaning.

\(^1\) Veale, Tony & Chen, Shanshan. *Unlocking the Latent Creativity of Orthographic Structure*. 

1.2 Utilising this feature for computational linguistic purposes

In the paper by Veale and Chen, this compositional quality of Chinese was exploited in order to extract semantic information about the English language. Quite simply, by taking an English word, which, when translated into Chinese produces a multi-logogram word, one can then break up this Chinese word into its parts, translate these individual parts back into English, and thus obtain a number of English words from the original single entry. From this series of words a new metaphorical interpretation might emerge; likewise one may realise new categorisations for the original English word. This, therefore, can reveal previously unknown dimensions of certain English words, and introduce us to an array of possibilities.

1.3 Developing this concept with the German Language

As the German language contains a considerable amount of composite words, the abovementioned approach regarding English and Chinese can also be applied to English and German, segmenting the compound German words into smaller words, as opposed to logograms.

The following diagram encapsulates the adopted strategy:
Figure 1: Sequences of steps to perform the translation
The arrows show the flow of the translation between the three components: the English text file, the virtual index and the German text file. The enumerated arrows describe the following:

(I) Connecting the English entry to its equivalent German entry
(II) Decomposing the German entry into sub-parts.
(III) Connecting The German sub-parts to their equivalent English entries

The objective was achieved by availing of the following language resources: EuroWordNet database; a code written by the project supervisor Dr Emms; and the code written by the author using C++ on a UNIX platform.

1.4 Why seek circumlocution?

In the world of search engines, a powerful one should involve searching topics based on their semantics, rather than the system that currently exists in the cyber world. Using the aforesaid methodology to mine a hidden interpretation, the gaps in current ontologies\(^1\) such as WordNet and Euro WordNet can be filled; or perhaps a balance can be created between the equivalent nodes, that are currently unbalanced in terms of content.

\(^{1}\)“an ontology is the set of semantic relations between concepts” - http://www.illc.uva.nl/EuroWordNet/objectives-ewn.html
Chapter 2

EuroWordNet

2.1 Introduction

In this chapter a history of the Lexical resource Euro WordNet and its structure will be given. Then, important features which have been employed in the project, such as the synsets with their ‘import/export’ format, Synonymy, the Equivalent Links, and the inter-lingual-index will be discussed.

2.2 What is EuroWordNet?

EuroWordNet is a multilingual lexical database resource based on the Princeton WordNet\(^1\). WordNet, also referred to as WordNet 1.5,

… is a large lexical database of English, developed under the direction of George A. Miller. Nouns, verbs, adjectives and adverbs are grouped into sets of cognitive synonyms.\(^2\)

It contains wordnets (hierarchies of words based on semantic relation) for several different European languages; English, German, Dutch, Spanish, Italian, French, Czech and Estonian. Both WordNet and EuroWordNet are lexical ontologies, the former solely for English, and the latter for all of the abovementioned languages.

A very important factor when discussing WordNet is the idea of the synset. A synset is a structure, made up of a group of words of the same part-of-speech, which can at times be used interchangeably. Synsets are connected together by relations that are

---


\(^2\) http://wordnet.princeton.edu/
lexical and conceptual-semantic in nature. An example of this would be the group of words \{car; auto; automobile; motorcar; machine\}.

![Diagram of synsets related to “car” in its first sense, in WordNet1.5.](image)

In the above diagram some semantic/conceptual relations are shown. These relations are explained as follows:

**Hypernym:**
A word ‘A’ is a hypernym of another word ‘B’ if ‘A’s’ meaning encompasses the meaning of ‘B’, that is, if ‘B’ is a kind of ‘A’. For example the word ‘furniture’ is a hypernym of the words ‘chair’ and ‘table’.

**Meronym:**
A meronym is a word that names a part of a larger whole; ‘brim’ and ‘crown’ are meronyms of ‘hat’.

**Hyponym:**
A term that denotes a subcategory of a more general class: “Chair “and “table” are hyponyms of “furniture”

---

2.3 Type of relations in EuroWordNet

There are several types of relation which are contained in the following two tables below. It is worth mentioning that the essential relation to the concept of EuroWordNet is synonymy.

Table 1: Language Internal Relations between synsets in EuroWordNet

<table>
<thead>
<tr>
<th>Relation Type</th>
<th>Parts of Speech</th>
<th>Labels</th>
<th>Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>NEAR_SYNONYM</td>
<td>N&lt;&gt;N, V&lt;&gt;V</td>
<td></td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>XPOS_NEAR_SYNONYM</td>
<td>N&lt;&gt;V, N&lt;&gt;AdjAdv, V&lt;&gt;AdjAdv</td>
<td></td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>HAS_HYPERONYM</td>
<td>N&lt;&gt;N, V&lt;&gt;V</td>
<td>dis, con</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>HAS_HYPONYM</td>
<td>N&lt;&gt;N, V&lt;&gt;V</td>
<td>dis</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>HAS_XPOS_HYPERONYM</td>
<td>N&lt;&gt;V, N&lt;&gt;AdjAdv, V&lt;&gt;AdjAdv</td>
<td></td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>HAS_XPOS_HYPONYM</td>
<td>N&lt;&gt;V, N&lt;&gt;AdjAdv, V&lt;&gt;AdjAdv, N&lt;&gt;N, AdjAdv&gt;V</td>
<td></td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>HAS_HOLONYM</td>
<td>N&lt;&gt;N</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>HAS_HOLO_PART</td>
<td>N&lt;&gt;N</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>HAS_HOLO_MEMBER</td>
<td>N&lt;&gt;N</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>HAS_HOLO_PORTION</td>
<td>N&lt;&gt;N</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>HAS_HOLO_MADEOF</td>
<td>N&lt;&gt;N</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>HAS_HOLO_LOCATION</td>
<td>N&lt;&gt;N</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>HAS_MERONYM</td>
<td>N&lt;&gt;N</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>HAS_MERO_PART</td>
<td>N&lt;&gt;N</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>HAS_MERO_MEMBER</td>
<td>N&lt;&gt;N</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>HAS_MERO_MADEOF</td>
<td>N&lt;&gt;N</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>HAS_MERO_LOCATION</td>
<td>N&lt;&gt;N</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>ANTONYM</td>
<td>N&lt;&gt;N, V&lt;&gt;V</td>
<td></td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>XPOS_ANTONYM</td>
<td>N&lt;&gt;N, V&lt;&gt;V</td>
<td></td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>CAUSES</td>
<td>V&lt;&gt;V, N&lt;&gt;N, V&lt;&gt;N, N&lt;&gt;AdjAdv, N&lt;&gt;AdjAdv</td>
<td>dis, con, non-f, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>HAS_SUBEVENT</td>
<td>V&lt;&gt;V, N&lt;&gt;N, V&lt;&gt;N, N&lt;&gt;AdjAdv</td>
<td>dis, con, non-f, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>IS_SUBEVENT</td>
<td>V&lt;&gt;V, N&lt;&gt;N, N&lt;&gt;N, V&lt;&gt;N</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>ROLE</td>
<td>N&lt;&gt;V, N&lt;&gt;N, AdjAdv&gt;N, AdjAdv&gt;V</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>ROLE_AGENT</td>
<td>N&lt;&gt;V, N&lt;&gt;N</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>ROLE_INSTRUMENT</td>
<td>N&lt;&gt;V, N&lt;&gt;N</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>ROLE_LOCATION</td>
<td>N&lt;&gt;V, N&lt;&gt;N, AdjAdv&gt;N, AdjAdv&gt;V</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>ROLE_DIRECTION</td>
<td>N&lt;&gt;V, N&lt;&gt;N, AdjAdv&gt;N, AdjAdv&gt;V</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>ROLE_SOURCE_DIRECTION</td>
<td>N&lt;&gt;V, N&lt;&gt;N, AdjAdv&gt;N, AdjAdv&gt;V</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>ROLE_TARGET_DIRECTION</td>
<td>N&lt;&gt;V, N&lt;&gt;N, AdjAdv&gt;N, AdjAdv&gt;V</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>ROLE_RESULT</td>
<td>N&lt;&gt;V, N&lt;&gt;N</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>ROLE_MANNER</td>
<td>AdjAdv&gt;N, AdjAdv&gt;V</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>INVOLVED</td>
<td>V&lt;&gt;N, N&lt;&gt;N, V&lt;&gt;AdjAdv, N&lt;&gt;AdjAdv</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>INVOLVED_AGENT</td>
<td>V&lt;&gt;N, N&lt;&gt;N</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>INVOLVED_INSTRUMENT</td>
<td>V&lt;&gt;N, N&lt;&gt;N</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>INVOLVED_SOURCE_DIRECTION</td>
<td>V&lt;&gt;N, N&lt;&gt;N, V&lt;&gt;AdjAdv, N&lt;&gt;AdjAdv</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>INVOLVED_RESULT</td>
<td>V&lt;&gt;N, N&lt;&gt;N, V&lt;&gt;AdjAdv, N&lt;&gt;AdjAdv</td>
<td>dis, con, rev, neg</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>CO_ROLE</td>
<td>N&lt;&gt;N</td>
<td>rev</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>CO_AGENT_PATIENT</td>
<td>N&lt;&gt;N</td>
<td>rev</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>CO_AGENT_INSTRUMENT</td>
<td>N&lt;&gt;N</td>
<td>rev</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>CO_AGENT_RESULT</td>
<td>N&lt;&gt;N</td>
<td>rev</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>CO_PATIENT_AGENT</td>
<td>N&lt;&gt;N</td>
<td>rev</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
<tr>
<td>CO_PATIENT_INSTRUMENT</td>
<td>N&lt;&gt;N</td>
<td>rev</td>
<td>Syn &lt;&gt;Syn</td>
</tr>
</tbody>
</table>

### Table 2: Language-Internal Relations between other data types in EuroWordNet

<table>
<thead>
<tr>
<th>Relation Type</th>
<th>Parts of Speech</th>
<th>Labels</th>
<th>Data Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>is_derived_from</td>
<td>N, V, AdjAdv (across all)</td>
<td>VA&lt;&gt;VA</td>
<td></td>
</tr>
<tr>
<td>has_derived</td>
<td>N, V, AdjAdv (across all)</td>
<td>VA&lt;&gt;VA</td>
<td></td>
</tr>
<tr>
<td>derivation</td>
<td>N, V, AdjAdv (across all)</td>
<td>VA&lt;&gt;VA</td>
<td></td>
</tr>
<tr>
<td>antonym</td>
<td>N&lt;&gt;N, V&lt;&gt;V, AdjAdv &lt;&gt; AdjAdv</td>
<td>VA&lt;&gt;VA</td>
<td></td>
</tr>
<tr>
<td>pertains_to</td>
<td>AdjAdv&gt;N, AdjAdv&gt;V</td>
<td>VA&lt;&gt;VA</td>
<td></td>
</tr>
<tr>
<td>is_pertained_to</td>
<td>N&gt;AdjAdv, V&gt;AdjAdv</td>
<td>VA&lt;&gt;VA</td>
<td></td>
</tr>
<tr>
<td>has_instance</td>
<td>N&gt;PN</td>
<td>Syn&gt;I</td>
<td></td>
</tr>
<tr>
<td>belongs_to_class</td>
<td>PN&gt;N</td>
<td>I&gt;Syn</td>
<td></td>
</tr>
</tbody>
</table>

2.4 Synonymy

This relation is central to the structure of EuroWordNet, and in effect, to the project. The translation from English to German and back to English is realised between synsets that are considered to have a synonymy relation.

2.5 EuroWordNet Import/Export format

EuroWordNet database files are organised using an import/export format. There are three types of format:
- Synsets
- ILI-records (These contain records of each synset’s equivalence relation)
- Top-Concepts and Domains

---

Given that it was principally the synset format that was relied upon in this project, it is sufficient to simply give an outline of that alone. With the synset format, one can import concepts for a language-specific wordnet. To clarify, an example of a synset can be found below.

```
0 @55718@ WORD_MEANING
  1 PART_OF_SPEECH "n"
  1 VARIANTS
    2 LITERAL "job"
      3 SENSE 2
      3 DEFINITION "what you should do for a living"
      3 EXTERNAL_INFO
        4 SOURCE_ID 1
        5 TEXT_KEY "08508615-n"
    2 LITERAL "work"
      3 SENSE 1
      3 STATUS "New"
      3 DEFINITION "what you do for a living"
      3 USAGE_LABELS
        4 USAGE_LABEL "sub"
          5 USAGE_LABEL_VALUE "Medicine"
        4 USAGE_LABEL "reg"
          5 USAGE_LABEL_VALUE "Informal"
        4 USAGE_LABEL "orig"
          5 USAGE_LABEL_VALUE "Latin"
      3 FEATURES
        4 FEATURE "connotation"
          5 FEATURE_VALUE "figurative"
        4 FEATURE "gender"
          5 FEATURE_VALUE "feminine"
        4 FEATURE "number"
          5 FEATURE_VALUE "singular"
    3 EXTERNAL_INFO
      4 CORPUS_ID 2
      5 FREQUENCY 920575
    4 SOURCE_ID 1
      5 TEXT_KEY "II.6.a"
    4 SOURCE_ID 3
      5 NUMBER_KEY 8008
    4 PAROLE_ID 36721
  1 INTERNAL_LINKS
    2 RELATION "has_hyponym"
      3 TARGET_CONCEPT
        4 PART_OF_SPEECH "n"
        4 LITERAL "lexicography"
        5 SENSE 9
    2 RELATION "has_hyperonym"
      3 TARGET_CONCEPT
```
The structure starts with the line “0 @55718@ WORD_MEANING”, which identifies the synset. At the next level (1) one can find information on:

- The part of speech: noun “n”, verb “v”, adjective, adverb (“AdjAdv” = Adjective or Adverb) or proper noun “pn”
- The variant, synset members or synonyms
- The language internal-relations (The relation between synsets within a wordnet)
- The equivalence relations\(^\text{1}\) (The relation between synsets across languages via ILI)

Two of the most important elements here, which play a chief role in how to implement the program when performing the translation, are variants and equivalence relations (EQ_LINKS). The literal and sense levels are inherent parts of each variant and are indispensable to the EuroWordNet program, and hence to this project. With regard to the equivalence relations, they are specified one by one by pointing to the type of relation and the target ILI-record. The ILI-record has, as part of the embedded information, either the file offset position or the ADD-ON id number. The former originates from the original WordNet1.5 and the latter from EuroWordNet. Each of them, together with the target-ILI and the variant, determine the type of the equivalence relation.

2.6 Inter-lingual-index

The Inter-lingual-index (ILI) contains records of all equivalence relations between synsets and WordNet1.5. Its purpose is to achieve an efficient mapping between languages. Synsets in wordnets that are linked to the same ILI-record, are considered to be equivalent across languages.

2.7 Conclusion

The above notions were briefly discussed in order to demonstrate key concepts which were employed in the program. In this chapter the key concepts of EuroWordNet that are of great relevance to the program were introduced and described. These concepts are: synsets, the synonymy relations between them, the concept of equivalence relations and the inter-lingual-index.
Chapter: 3

Use and Implementation

3.1 Introduction

The aim of this chapter is to describe and discuss the software that has been developed. This entails outlining the general algorithm. Moreover, it will give a description of the code written by the author, to perform the translation of English entries (using English EWN_entries\textsuperscript{1} from the Euro Wordnet database) into German (using German EWN_entries from the Euro WordNet database) and then back again from the sub-parts of the German entries into the equivalent English EWN_entries.

3.2 Description of the Software

The programming language used to develop the software is C++ on a UNIX platform. The software consists of two parts: part one was written by Dr. Martin Emms, supervisor of this project, and part two by the author. Firstly, part one will be discussed.

\textsuperscript{1} An EWN_entry is a synset
The first part can be divided into two sets presented in the following table:

Table 3: Software provided by the supervisor

<table>
<thead>
<tr>
<th>Classes</th>
<th>Database files containing entries</th>
</tr>
</thead>
<tbody>
<tr>
<td>Build_eqlink_index.cpp</td>
<td>Wn_15_adjectives.ewn</td>
</tr>
<tr>
<td>StringIndex.cpp</td>
<td>Wn_15_adverbs.ewn</td>
</tr>
<tr>
<td>Ewn.cpp</td>
<td>Wn_15_nouns.ewn</td>
</tr>
<tr>
<td>ILL_index.cpp</td>
<td>Wn_15_verbs.ew</td>
</tr>
<tr>
<td>Ger_to_eng.cpp</td>
<td>Wn_de.ew</td>
</tr>
<tr>
<td>Wn_15_nouns_index</td>
<td>Wn_15_nouns_index</td>
</tr>
<tr>
<td>Wn_15_adjectives_index</td>
<td></td>
</tr>
<tr>
<td>Wn_15_adverbs_index</td>
<td></td>
</tr>
<tr>
<td>Wn_15_nouns_index</td>
<td></td>
</tr>
<tr>
<td>Wn_15_verbs_index</td>
<td></td>
</tr>
<tr>
<td>Wn_de_index</td>
<td></td>
</tr>
<tr>
<td>Wn_english_addition_index</td>
<td></td>
</tr>
<tr>
<td>Ewn_test.cpp</td>
<td></td>
</tr>
<tr>
<td>ILL_index_test.cpp</td>
<td></td>
</tr>
</tbody>
</table>

The second part contains the software developed by the author and is presented as follows:

- Build_string_index.cpp
- Wn_15_adjectives_var_index
- Wn_15_adverbs_var_index
- Wn_15_nouns_var_index
- Wn_15_verbs_var_index
- Wn_de_var_index
- Wn_english_addition_var_index
- Entry_to_Entry.cpp
3.3 The Logic behind the Design

As mentioned before, the inter-lingual-index is the backbone of concept nodes (synsets) in the EWN ontology. These concept nodes contain inner structures called EQ-Links that serve to connect these nodes to the inter-lingual-index and therefore to other equivalent concept nodes. Every synset that exists in the database has to have an EQ_Link; an example of the structure of an EQ_Link is illustrated in the following excerpt from a database file and for the purpose of clarity has been highlighted in bold in order to distinguish it from the rest of the synset:

```
0 @55718@ WORD_MEANING
1 PART_OF_SPEECH "n"
1 VARIANTS
  2 LITERAL "job"
    3 SENSE 2
      3 DEFINITION "what you should do for a living"
    3 EXTERNAL_INFO
      4 SOURCE_ID 1
        5 TEXT_KEY "08508615-n"
  2 LITERAL "work"
    3 SENSE 1
      3 STATUS "New"
      3 DEFINITION "what you do for a living"

3 USAGE_LABELS
```
USAGE_LABEL "sub"
  USAGE_LABEL_VALUE "Medicine"
USAGE_LABEL "reg"
  USAGE_LABEL_VALUE "Informal"
USAGE_LABEL "orig"
  USAGE_LABEL_VALUE "Latin"
FEATURES
  FEATURE "connotation"
    FEATURE_VALUE "figurative"
  FEATURE "gender"
    FEATURE_VALUE "feminine"
  FEATURE "number"
    FEATURE_VALUE "singular"
EXTERNAL_INFO
  CORPUS_ID 2
  FREQUENCY 920575
  SOURCE_ID 1
  TEXT_KEY "II.6.a"
  SOURCE_ID 3
  NUMBER_KEY 8008
  PAROLE_ID 36721
INTERNAL_LINKS
  RELATION "has_hyponym"
    TARGET_CONCEPT
      PART_OF_SPEECH "n"
      LITERAL "lexicography"
      SENSE 9
  RELATION "has_hyperonym"
    TARGET_CONCEPT
      PART_OF_SPEECH "n"
      LITERAL "activity"
      SENSE 3
EQ_LINKS
  EQ_RELATION "eq_has_hyperonym"
    TARGET_ILI
      PART_OF_SPEECH "n"
      WORDNET_OFFSET 8508615
  EQ_RELATION "eq_near_synonym"
    TARGET_ILI
      PART_OF_SPEECH "n"
      WORDNET_OFFSET 2861550
  EQ_RELATION "eq_generalization"
    TARGET_ILI
      PART_OF_SPEECH "n"
      ADD_ON_ID 8543

The function of the EQ_Link is to direct the synset that contains it to the position of
the synset with the equivalent meaning in the database file of the other language. A
sequence of items is contained within an EQ_Link; each item consisting of an
EQ_RELATION and a TARGET_ILI.

A crucial factor, already briefly mentioned, which had to be taken into account
when designing the software, was the position of the synset in the database file. The
first thing that must be done when accessing a synset, is to obtain its position in the
file. Once the position is obtained the synset content can be retrieved and hence the
translation is enabled. This gave rise to implementing an “inverted index” approach
which plays a similar role to that of the inverted index at the end of a book. The
inverted index approach serves two purposes:

1- Having created an inverted index for an EQ_link, one is able to retrieve the
position of the synset that contains that EQ_Link in the database file via the inverted
index entry. Thus, entries in the inverted index enable translating from one particular
language to another.
2- After translating from English to German and retrieving the composite German
lexeme, the inverted index approach enables us to look up the sub-parts that once
composed the German lexeme in the German database file, to see if they exist as
separate words in their own right.

3.4 Algorithm

Here are the steps of the algorithm designed by the author:
1) Retrieving English entries from English WordNet database files.
2) Obtaining the translation of the variants of English entries in German.
3) Decomposing the resulting compound German variants into several different
sub-parts.
4) These sub-parts are then looked up in the German database file.
5) If they exist in the German database file, they are then translated back into
English.
3.5 Classes’ Specifications

Classes were defined and implemented to represent several objects which were manipulated to perform the translation from English to German and back again to English. These classes are:

• Ewn.cpp: This is the top-level class and it represents a Euro WordNet entry. The class has a break-down that corresponds to the format for WordNet entries which is referred to with the ‘import/export’ format.

Things to consider:

1. The structures: PART_OF_SPEECH, RELATION (under INTERNAL_LINKS) and EQ_RELATION (under EQ_LINKS) have, in each case, a finite list of possible alternative values. Accordingly, in the header of this class they are mapped into the enumerated types: PartOfSpeech, InternalReln and EqReln. These enumerated types assign a number to each of the possibilities that they contain.

Here are the functions, mapping between the numbers in this enumeration, and the strings which they stand for:

```cpp
string pos(PartOfSpeech);
PartOfSpeech pos_from_string(string p_string);

string int_rel_str(InternalReln);
InternalReln rel_from_string(string r_string);

string eq_rel_str(EqReln r);
EqReln eq_rel_from_string(string rel_string);
```

2. The part of the EQ_Link which is called TARGET_ILI contains - according to the documentation of Euro WordNet - either a WORDNET_OFFSET part or an ADD_ON_ID part. This had to
be taken into consideration when it came to defining the EqLink in
the Ewn.cpp class. The class has an attribute: vector<EqLink>
equivalence_links.
Where EqLink is:

```
struct EqLink {
    EqReln eq_relation;
    struct {
        PartOfSpeech part_of_speech;
        unsigned int wordnet_offset;
        unsigned int add_on_id;
        bool with_offset;
    } target;
};
```

The design decision taken here is to set:
with_offset == true when the WORDNET_OFFSET part is active.
with_offset == false when the ADD_ON_ID part is active.

3. The class has also a member file_pos of the type
streampos. Having created an EWN_entry object from lines of
text in a file it is then used to register the position in the file at
which the lines of a EWN synset begin. This notion of current file
position is part of the C++ file processing classes.

```
file_pos = f.tellg(): this is how you get the file
position from a stream f.
f.seekg() = file_pos: this is how you set the file position
from a stream f.
```

File positions are also referred to as file offsets.
4. Below are parts of the synset which weren’t taken into consideration here as their absence doesn’t compromise our objective:

FEATURES
REVERSED
EXTERNA_INFO
SOURCE_ID
TEXT_KEY

5. In creating an EWN_entry object a parsing mechanism has to be implemented. The key functions of the parsing code are:

```cpp
bool get_next_EWN_entry( ifstream& f, EWN_entry& e);
```

This function tries to set an object `e` of type EWN_entry from the next lines of text found in `ifstream f`. If it succeeds to do this, it returns true otherwise it returns false. It will also fail if the last entry has already been read out of the file. This function implements the parsing code. The idea of parsing is that each EWN_entry has the structure of a tree, with an explicitly indicated opening bracket which can be looked at abstractly in the following form:
where N is a number that serves as a tag which indicates the depth of bracketing before the label XXX.

The tags also indicate the presence of closing brackets. This can be observed in the following abstract notation. After an opening:

N XXX...

One detects that this is complete when encountering:

M YYY..

where M is a different tag and YYY a different label with M <= N (or the end of a file).

The above strategy can be more vividly understood in the context of the following analogy:

Since an entry structure in the EWN would look like this:

```
0 @55718@ WORD_MEANING
1 PART_OF_SPEECH "n"
1 VARIANTS
  2 LITERAL "job"
  3 SENSE 2
  3 DEFINITION "what you should do for..."
  3 EXTERNAL_INFO
  4 SOURC_ES_ID 1
  5 TEXT_KEY "08508615-n"
2 LITERAL "work"
  3 SENSE 1
  3 STATUS "New"
  3 DEFINITION "what you do for a..."
  3 USAGE_LABELS
1 INTERNAL_LINKS
```
This shows that when constructing the parsing technique, it should parse by pushing the opening brackets into a stack and popping the closing brackets. Therefore the indent number which is associated with popping has the purpose of finishing the structure building.

Below is an example of this:

<table>
<thead>
<tr>
<th>line num</th>
<th>stack</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>(1 POS)</td>
</tr>
<tr>
<td>3</td>
<td>(1 VARS)</td>
</tr>
<tr>
<td>4</td>
<td>(1 VARS)(2 LIT) fill placeholder Variant v.literal</td>
</tr>
<tr>
<td>5</td>
<td>(1 VARS)(2 LIT) fix v.sense_id</td>
</tr>
<tr>
<td>6</td>
<td>(1 VARS)(2 LIT)(3 DEF) fix v.def</td>
</tr>
<tr>
<td>7</td>
<td>pop (3 DEF) and (2 LIT) push v on variants then push (2 LIT)</td>
</tr>
<tr>
<td>8</td>
<td>(1 VARS)(2 LIT) fill v.literal</td>
</tr>
<tr>
<td>9</td>
<td>(1 VARS)(2 LIT) fill v.sense_id</td>
</tr>
<tr>
<td>10</td>
<td>pop (2 LIT) push v on variants pop (1 VARS) do any variants tidy</td>
</tr>
</tbody>
</table>

   then push (1 INT_LINKS)
   (1 INT_LINKS)

: 6. bool get_next_EWN_entry (ifstream& f, streampos fpos, EWN_entry& e).

The difference between this function and the above one is that this version sets the file position to fpos before attempting to parse an EWN_entry. It therefore facilitates going from file positions saved in an index to the entries at those file positions.
7. The function ‘get_next_EWN_entry(..)’ has a local variable –
‘vector <StackItem> st’ which is used to trace the progress
through the parsing process.

8. The functions do_open(..) and do_close(..) are called in
the process of building an EWN_entry. The first function is called
when a bracket is opened, and the second when a bracket is closed,
after constructing the embedded information.

The attributes of an EWN_entry differ in their structure.
Therefore sometimes the information for structure building is all
on one line (e.g. PART_OF_SPEECH). When this is the case
do_open(..) sets the relevant parts of the EWN_entry.

On the other hand, sometimes the information for structure
building is distributed over several lines.
Consequently, all list-type information such as VARIANTS,
INTERNAL_LINKS, EQ_LINKS, and in this case
do_open(..) sets parts of place-holder variables such as:

```cpp
Variant v
InternalLink int1
EqLink eql
```

Then the function do_close(..) uses the above variables to do
the appropriate updates of the EWN_entry.

The operator “>>” of the istream class plays the chief
role in segmenting a line of characters into distinct parts. It is used
to convert a string into a stream and then parts are scanned (string,
ints, etc) out of it, into appropriately typed variables.

- **ILI_index.cpp**: The header (ILI_index.h) defines the inverted index
  of an EqLink. The inverted-index maps an EqLink object (eql) to the
  offset of the EWN_entry that contains it. The implemented index
  functions in ILI_index.cpp are based on the Berkeley database
  library. Two important instructions have to be discussed:
- The first concerns the updating of the offsets stored for an
EqLink object (eql), assuming the offsets are given as a vector:

  key: key.data = &eql
  data: make a dynamically allocated int array into which
  the vector is copied.
  data.data = address of dynamic array

- The second concerns retrieving the offsets stored for an
EqLink

  object (eql):
  key: key.data = &eql
  data.data points to a sequence of 4 byte integers: b0 b1 b2 b3.

  For alignment first copy byte-by-byte to an int variable
  and then store in offsets array.

The code used to search for an EqLink object (eql), in an already
built-up inverted-index (e.g.; in the Wn_nouns_index) goes as
follows:

ILI_index english_db;

english_db.open(true,"Wn_nouns_index") vector<unsigned int> offsets;
get_offsets(eql,offsets);

- The program build_eqlink_index.cpp makes use of the above
code to build the inverted-index files from the EWN database files. The
following table shows the particular EWN database files i.e.(nouns,
adjectives, verbs, etc) files and their inverted-index files:
Table 4: Database files and their inverted indices

<table>
<thead>
<tr>
<th>EWN_Data/</th>
<th>EWN_Software/</th>
</tr>
</thead>
<tbody>
<tr>
<td>wn_15_adjectives.ewn</td>
<td>wn_15_adjectives_index</td>
</tr>
<tr>
<td>wn_15_adverbs.ewn</td>
<td>wn_15_adverbs_index</td>
</tr>
<tr>
<td>wn_15_nouns.ewn</td>
<td>wn_15_nouns_index</td>
</tr>
<tr>
<td>wn_15_verbs.ewn</td>
<td>wn_15_verbs_index</td>
</tr>
<tr>
<td>wn_english_addition.ewn</td>
<td>wn_english_addition_index</td>
</tr>
<tr>
<td>wn_de.ewn</td>
<td>wn_de_index</td>
</tr>
</tbody>
</table>

- The `StringIndex.cpp` is implemented in the same way as the `ILI_index.cpp` with the exception that it deals with embedded information, under the heading of VARIANTS. More precisely, it deals with LITERALS.
- The `build_string_index.cpp` program creates inverted indices for LITERALS. It is the same as the `build_eqlink_index.cpp`, except that rather than dealing with EQ_Links it deals with LITERALS.
- The `ILI_index_test.cpp` is a program for testing an already built index against a file. The command line to compile it is:

  make ILI_index_test

  To run it:

  ILI_index_test INDEX_NAME SOURCE_FILE_NAME

  One should see nothing, unless an EqLink is found in the source file for which no offset information is saved in the index.

- The `Ger_to_eng.cpp` program performs the translation from a German EWN_entry to an English EWN_entry. It shows how inverted-indices of EqLinks are used to translate entries. In the program two instances `english_e` and `german_e` are declared to be of type EWN_entry.
After the inverted-indices for the EqLinks have been created, they are used to give the positions of the EWN entries, containing the EqLinks. The following illustration demonstrates the steps between an instance ‘eql’ of type EqLink, the inverted-index and the file position of the entry.

‘eql’ \(\rightarrow\) inverted-index \(\rightarrow\) positions of EWN entry containing eql

An instance of type EWN_entry, english_e is declared in the class. If we assume that english_e in an English EWN database file is at E_offset, we can consider the following:

0 0575750 SOME_SYNSET_NAME
1 PART_OF_SPEECH "n"
1 VARIANTS
2 LITERAL "job"
:
1 INTERNAL_LINKS
:
1 EQ_LINKS
2 EQ_RELATION "eq_has_synonym"
3 TARGET_ILI
4 PART_OF_SPEECH "n"
4 WORDNET_OFFSET 564545

If ‘eql’ is an EqLink that is parsed from this entry, then E_OFFSET should be the value that english_index returns on eql:

E OFFSET in english_index(eql)
Likewise, assuming that an instance of type EWN_entry is `german_e` in the German EWN database file at offset `G_OFFSET`, it follows that:

```
0 @57575@ SOME_SYNSET_NAME
   1 PART_OF_SPEECH "n"
   1 VARIANTS
      2 LITERAL "beschaeftigung"
   :
   1 INTERNAL_LINKS
   :
   1 EQ_LINKS
      2 EQ_RELATION "eq_has_synonym"
      3 TARGET_ILI
         4 PART_OF_SPEECH "n"
         4 WORDNET_OFFSET 564545
```

If eql is an EqLink that is parsed from this entry, then `G_OFFSET` should be the value that `german_index` returns on eql:

```
G_OFFSET in german_index(eql)
```

The function `print_equiv_from_an_index(source_e, int, target_index, target_ewn)` performs the translation. The parameters are passed here:

- `source_e` is the source-entry from the EWN database files
- `target_ewn` is a target EWN database file
- `target_index` is its index
The code can translate from the entry source_e in source_ewn (EWN source database files) to the entry target_e in target_ewn in the following way:

- Get any ‘EqLink eql’ inside source_e
- Get any TARGET_OFFSET in target_index using ‘eql’
- Retrieve target entry target_e in target_ewn at offset TARGET_OFFSET

The Entry_to_Entry.cpp program performs the abovementioned algorithm. Next an account of the implemented functions will be given.

- The function bool get_equiv_Ent_FromIndex(...) realise the cross language binding (linking) between a source entry in a text file of a particular language and its equivalent target entry in a text file of another language. The central part of the function is the invocation of the get_offsets(...) on the target inverted index object. This employs the equivalence relation (EQ_LINKS) to retrieve the offsets of the target synsets.
- The function ge_split(...) divides German entries into their sub-parts.
- The function get_splits_offsets(..) retrieves the offsets of the German sub-parts. Since a sub-part might have more than one offset, the object Pair_and_offsets had to be introduced.

```
Class Pair_and_offsets {
    Public;
    string literal_first;
    string  literal_second;
    vector<unsigned int> first_offsets;
    vector<unsigned int> second_offsets;
    void print(void);
}
```

The function then returns this object.
3.6 The logic of the program

To conclude, here is a sequence diagram that illustrates the above:

Figure 3: Sequence Diagram 1
Figure 4: Sequence diagram 2.

```
: Entry To Entry : EWN entry : ILI index : EWN target_entry : literals : Vector

equivalent_links = equivalence_links()
target_offsets = get_offsets()
target_entry = get_next_EWN_entry()
target_entry_variants = variants()
push_back()
```

For each equivalent link
For each target offset
For each target entry variant
Chapter 4

4.1 Limitations

There are many German words that conform to a certain grammatical rule, which allows them to combine and form one word. This rule demands an insertion of infixes such as ‘e’, ‘en’ and ‘s’. The rule was not taken into consideration when implementing the program, which unfortunately meant that a reduced number of translations were retrieved.

Some synsets have a great number of variants that initially resulted in slowing down the running of the program. To enable the program to run faster, only synsets that contain 10 variants at maximum were considered. Another factor that has an impact on the amount of information been retrieved is the unbalanced nature of EuroWordNet ontology.

Moreover, due to the time limitation the translation was realised only between the English nouns text file and German. The other files are adjectives, verbs and adverbs.

4.2 Conclusion

The translation was achieved between English entries and German entries, then the composite German entries were decomposed and translated back into English. The results were intriguing and it was surprising to realise the richness of the English language, by the amount of metaphors and paraphrases that were discovered. Although, as outlined above, the study is by no means complete, it demonstrates the value of pursuing further statistical investigation into how many meaningful words we can retrieve by decomposing composite German words. Finally, this project will hopefully provide a basis for future work investigating the similarity or distance between languages, and thus enrich our knowledge of the etymological basis of the languages in question.
References


- http://wordnet.princeton.edu/