A tool for rapid manual translation
by
Magnus Nordström and Paul Pettersson
A tool for rapid manual translation

by

Magnus Nordström and Paul Pettersson
dilemma@sics.se

Swedish Institute of Computer Science
Box 1263, S-164 28 Kista, Sweden

October 11, 1993

Abstract

There have been several attempts to realize the idea of a fully automatic translation system for text translation to replace human translators. By contrast, little work has been put into building tools to aid human translators.

This report describes the ideas behind such a tool. The tool is intended to aid human translators in achieving higher productivity and better quality, by presenting terminological information extracted from previous translations. The report documents the implementation and evaluation of a prototype. The prototype has been demonstrated to and used by professional translators with promising results.

KEYWORDS: translation aid, evaluation of a prototype
# Contents

Abstract

Table of contents i

1 Introduction 1

2 Translation 2
  2.1 Automatic translation 2
  2.2 Translation aid systems 3
  2.3 The translation process 3

3 Methodology 5
  3.1 Prototyping and Programming environment 5
  3.2 Object oriented design 6
  3.3 Baseline 6
  3.4 Project design development history 7

4 Basic concepts 9
  4.1 Working with Dilemma 9
  4.2 String Alignment 9
  4.3 Definitions of elements 10
  4.4 Finding occurrences of a word 12
  4.5 Translating words 13
  4.6 Morphosyntactic analysis 16

5 Designing the application - stages 16
  5.1 Interactive tool with hierarchical levels 17
  5.2 Translating words 20
  5.3 Batch analyses 23

6 Implementation Issues 25
  6.1 Interactive tool with hierarchical levels 25
  6.2 Translating words 26
  6.3 Batch analyses 27

7 Evaluation 28

8 Discussion 30

Appendix A Design Guidelines by Microsoft 32
Appendix B An example of Dilemma's ini-file 36
Appendix C BNF of the Meta file 39

References 41
1 Introduction

This report documents the implementation of a tool for rapid manual translation and the investigation of required methods.

The contributions of this work are: to describe methods to aid manual translation without using linguistic analysis, to describe statistical methods to associate relevant words (translations) in a bi-lingual corpora, and to demonstrate these methods in a running prototype together with a word processor in a graphical environment.

The concept takes advantage of the fact that translators have access to substantial corpora (so called bi-texts). These can be used to improve the characteristics, such as speed and quality, of the translation process.

The report is organized as follows: Section 2 discuss the translation process; Section 3 documents the methodology used in the project; Section 4 defines the necessary concepts; Section 5 documents the design of the application; Section 6 provides details concerning implementation issues; Section 7 provides evaluation. We end with some conclusions.
2 Translation

The naive view of text translation is that it involves taking the source text, extracting its meaning, and then producing the destination text, which is in another language. In reality, the nature of human language hinders the application of this intuitively pleasing model. Firstly, language is a tool in use, and not an object to study in isolation, or in other words, text in itself arguably has no single well-determined meaning separated from its producer, or its intended or actual use. Secondly, the process of extracting a meaning from a text - semantic analysis - is not a problem that is even close to a solution. Indeed, even the representation of such a meaning is not even close to being determined in any generally useful way. Thirdly, for any intended meaning, there is close to an infinite number of ways of expressing it in any language [Karlsgren et al 1993]. In short, every text has many meanings, every meaning has many texts, and we do not know what meaning is or how to get at it.

2.1 Automatic Translation

However, in actuality, translation seems to work in practice. People translate large amounts of documents without running into the sort of theoretical problems of the kind that the pessimist might expect. And not only do translators actually translate, they do it without any great difficulty. In fact, translators complain that many text types are dull and repetitive to work with, and that the work is mechanical, making translation sound like a prime candidate for automatization. And translation has been a target for computerization since the fifties. Various projects have been initiated with various degrees of success, or rather, various degrees of failure. Machine translation has not made the impact on the translation community that machine translation researchers have hoped for and still hope for. "History provides no better example of the improper use of computers than machine translation" in Martin Kay's words [Kay 1980]. The reason for this is mainly that the machine translation field has strived towards the wrong goals, most notably to building a completely automatic translation system.

Relaxing the Specification

From the theoretical standing point outlined above, building an automatic translation system from a human language to another is clearly a technologically risky project. FAHQGPM - Fully Automatic High Quality General Purpose Machine Translation - is not close to being a reality. In practice, to build useful tools, the way is to relax some of the specifications of the FAHQGPM project. This can be done in several ways. By relaxing the GP - general purpose goal - we get highly specific systems, which may be successful in their domain, if the knowledge structure of the domain is restricted enough for formalization. By relaxing the HQ - High Quality goal - we get systems that produce crude translations, which may be good enough for a specific purpose. And finally, by relaxing the FA - Fully Automatic goal, we may get systems which produce raw translations for further manual work.
2.2 Translation Aid Systems

In the Dilemma project, which was prompted by the needs of legal translation, where quality cannot be compromised, and the text is varied enough to be called general purpose, we have relaxed the M - machine - part of the goal, by building not a translation system at all, but a translation aid system. This is a tool in the spirit of the Translator's Amanuensis that Kay outlined in his paper, that is built to magnify human productivity, and "... by taking over what is mechanical and routine, it frees human beings for what is essentially human." [Kay 1980].

2.3 The Translation Process

Producing text is a smaller proportion of the translation process than may be realized at first. About the same proportion of work will need to be put in each of the following stages:

- preparation
- text production
- proofreading

which is reminiscent of the person-time breakdown of software development [Brooks 1975]. Each of the stages has its specific problems, but some of the work steps are similar throughout. In the Dilemma project we have built a tool that will be useful through the entire process, in all the three stages.

The Data At Hand - Bi-texts

The data to be used are previously translated texts in the same domain. We call these - using Pierre Isabelle's term [Isabelle 1992] - bi-texts. The general idea is to extract terminological information from these data, in order to maintain consistency in translation which often, e.g. in technical and legal text, is a quality criterion and to reduce the decision load of the translator, which always is a productivity criterion. The previously translated texts need to be aligned element by element on some specified hierarchical level, so that correspondences can be extracted. What the appropriate hierarchical level - phrase, clause, sentence, paragraph, etc - is, is domain and text specific to some degree.

The Necessary Functionality

Typical questions that a translator has when producing text, in the middle phase of the translation process, are

- how a word or phrase has been used in previous contexts, and
- how a word or phrase has been translated in previous contexts.
These are the questions Dilemma is designed to answer. These obviously occurring questions also have corollaries in the other process phases.

In the text preparation phase, a translator scans through the text for words and phrases that need to be looked up, resolved, or treated in any way. The two questions above are obviously relevant for this work phase as well.

In the proofreading phase, a translator or an editor reads the text to correct errors or inconsistencies. One of the obvious problems of ill-translated text are terminological misses, and the way to help investigate suspicious cases are obviously by providing answers to the two questions above.

The first step is to provide the translator with a look-up tool that will enable browsing through the bi-text and looking at pairwise text elements, picked out by a string search. The translator can use the tool to check for conventions of usage that are unfamiliar or unexpected.

The second step is to analyze the text elements and to try extract the likely translation of a search word. If the text elements are aligned correctly enough, this can be done with some degree of reliability, and the translator is given a list of candidate translations to choose from.

A third step is a natural extension when working in the preparation phase. The main goal when preparing a text and a translator for translation is to provide useful word and phrase lists that are text- and domain-specific. Therefore, a natural extension of the interactive word and phrase look-up tool is to batch produce lists of translations for the translator to use in the production phase.

Our objective has been to develop a interactive tool that will increase the productivity of the human translator; especially in case of larger textmasses. Dilemma does this by enabling translators to recycle earlier translations of words and phrases with small effort. This lessens the cognitive load of translators to a considerable degree. Moreover, Dilemma improves the quality of the translation, by facilitating the process of preserving the consistency between translated texts, even in cases where texts are written by several different translators. Dilemma does not provide all the tools a translator needs, and not even all the tools the translators are promised by visionary research projects, but Dilemma does provide some of the services a translator needs, and it does this in a way that is designed to fit in with the translation process as it is today.
3 Methodology

3.1 Prototyping and Programming environment

We found that the most natural model for us to use for software development was prototyping. We had several reasons for this conclusion. First, we wanted to develop the system rapidly in order to perform user experiments with the prototype.

Secondly, since we were not convinced what the graphical interface should look like to be effective, we had to be prepared for continuous changes in the graphical interface. In fact, we could not find an existing system that provided and presented services we were about to implement. Therefore, we hoped that the end-users would give us enough feedback to improve the interface.

Thirdly, we didn't know how much functionality we would be able to design and implement within the limited time. Fortunately, the (independent) phases in the project encouraged us even more to use prototyping and to extend the functionality of the tool in increments.

We chose to use Microsoft Windows 3.1 as the operating system for Dilemma. The reason for this choice is that many translators use wordprocessors (for example Word Perfect) that are written for Microsoft Windows 3.1.

Microsoft Windows 3.1 is very common in PC environments, but far from an ideal operating system when to implement applications. For one thing, Microsoft Windows 3.1 is not an operating system. It is an application running in another operating system; MS-DOS (MicroSoft Disk Operating System). We had some problems in implementing background processes due to the inherent limitations of MS-DOS. An serious alternative was OS/2 2.1, a true 32-bit, time sharing operating system. Even though it is possible to execute applications written for Microsoft Windows in OS/2 2.1, we chose Microsoft Windows instead of OS/2 2.1. The reason was that we found it safer to use the same environment during the development as for the prototype.

Our next step was to find a programming environment for Microsoft Windows. We wanted to use a programming environment which permitted us to change the presentation without laborious work. Furthermore, we wanted the environment to provide an abstraction from low-level programming. One candidate was LPA Prolog for Windows. Prolog is a language well suited for prototyping and LPA Prolog for Windows also supports graphical programming at a high level. Prolog is also definitely a high-level language, which would let us to concentrate on the problem and not on irrelevant programming details.

Unfortunately we could not wait for the latest release of LPA Prolog for Windows, so we had to look for alternative systems. Eventually we decided to use Microsoft Visual C++ for Windows. We believed that the class library supported by the system and the tool included in the system for drawing graphical interfaces would help us to implement Dilemma rapidly. This was only partly true. During the project we discovered that minor changes in the interface of Dilemma were sometimes difficult to accomplish.

Although we had chosen C++, we did use (SICStus) Prolog to implement prototypes in order to do preliminaries and for finding answers to certain problems. We found Prolog very suitable
for this kind of tasks and increased our productivity in the project. We believe that Prolog could have increased the productivity even more by using LPA Prolog exclusively throughout the project, instead of C++.

3.2 Object oriented design

A object oriented language like C++ is, of course, well-suited for object oriented design and programming. The idea of describing a system as a set of interacting objects is very convenient; especially if the objects are easy to identify. We found that one of the biggest problems in using a object oriented design was to identify these objects. But at the same time, the objects identified were validated during the project. Services added to a specific object often revealed if the object was a natural member in the system. It occurred several times that services where transferred between different objects and that objects were given new names. The system become very dynamic and we found this property, in most cases, very positive.

Object oriented design has another beneficial property; the system becomes modularized. This implies that the system is easy to maintain in the future. A modular design makes the system more portable to other environments. We have not prioritized the portability, but still, many classes can be directly recompiled to other operating systems with only small changes. The classes concerning the graphical issues are, of course, not directly portable. The reason is that we have used classes provided by the Microsoft Foundation Classes and they are highly specific for Microsoft Windows. Therefore, all code concerning the presentation of the data must be rewritten, if the system is to be ported to another environment.

Another effect of using object-oriented design was that we could separate the tasks into independent problems. For example, we found a clear distinction between presentation and functionality. By identifying the objects in the independent problems and the interfaces between them, the prototype could be developed rapidly. One factor that contributed to this effect, was the efficient communication we had during the project. The internal communication in the project has shown to be a crucial factor to the development time in a project [Brooks 1975] and this project was not an exception.

3.3 Baseline

During the project we had several discussions about design and representation. Our strategy during the project can be compared to a riskdriven model and the prototype has been developed incremently with continuous evaluations.

When the project started, there existed four modules which where developed by SICS and Scandface. They constitute a preliminary study to this project. We will now describe those modules briefly.
• \textit{Para} takes two corpora as input. These two corpora are analyzed with a method which is a modified version of the string-alignment algorithm described in section 4.2. \textit{Para} produces (in a external file) a description how these texts are aligned. In addition, numbers are presented to the user which represent the resemblance between the texts.

• \textit{Kolla} has a similar functionality to \textit{Para}, but provides information about assumed missing or mismatched text fragments. Additionally, \textit{Kolla} gives enough information to localize these fragments.

• \textit{Dil} is a modified version of \textit{Para} which gives extended information about the corpus structure. \textit{Dil} presents the hierarchical levels in the corpus in addition to the same information produced by \textit{Para}.

• \textit{Dilemma} is a preliminary study to the prototype with the same name. Given two corpus with numbered sentences and a search pattern, \textit{Dilemma} produces a list of words, ordered by frequency. Every word has been found in one or more sentences that corresponds to the sentences where the given search pattern was found.

\textit{Para}, \textit{Kolla} and \textit{Dil} have been developed by \textit{Scandface} and were implemented in \textit{muLisp}. \textit{Dilemma} has been developed by \textit{SICS} and was implemented in \textit{C}.

3.4 Overview of the phases in the project

We will now describe the different phases of the project. Each phase had a corresponding task(s) and what follows is a summary of how we achieved the task(s) and some of the difficulties we encountered. During the project, we found several areas that must be further investigated to improve the properties of the prototype (see discussion in section 8). However, the limited time did not allow us to do these further studies.

\textit{First step: Pairwise Browsing Through Bittext Elements}

In the first phase we solved the problem of finding text fragments at a specific hierarchical level (a typical hierarchical level can for example correspond to sentences; see section 4.3 for more information) in the source text which included a given search pattern and to present the corresponding translations (of those text fragments). In the beginning we had a fixed number of hierarchical levels (document, paragraph and phrase level). It was our intension to generalize the numbers of hierarchical levels later on in the project.

\textit{Dilemma} receives as input a bitext on the form of a source language text file, a destination language text file and an alignment file produced by the alignment program \textit{PARA}. In order to handle the output from \textit{Para}, we wrote a BNF which defined the structure of the output produced
by Para. The syntax was quite different from the previous output and Scandface took part of this definition and modified Para according to our specification. The graphical interface was in this stage rather primitive and clearly revealed our assumption about the hierarchical levels (fig 3.4.1).

Later on in the project we slightly modified the BNF in order to save space and to accept a general number of hierarchical levels. In addition, we improved the search process in the data produced by Para (see section 6.1)

Second step: Word Translation

The next phase was concerned word translation. We started to investigate what general properties a word and its corresponding translation had in common in a corpora. We found at least three properties; number of occurrences in the corpus; relative position within a hierarchical section (for example, in a paragraph) and the length of the word.

We defined a statistical method that, hopefully, would find a acceptable translation in the target text, given a word in the source text. The method should at least find a word in the target text that is, in some sense, relevant to the given word.

First, we believed that we needed a list of words that we knew a priori were unconnected to the given word (for example, words like "this"," the","a" etc) in order to increase the performance of the method. To examine if the assumption where true, we implemented the method in a Prolog program. This program took a perfectly aligned bi-text as input and presented for each word in the source text a possible translation. We found that without using any kind of ad hoc list, the method succeeded in ninety percent of the cases. We hoped that we later on could balance the factors in our method to increase the performance even more and therefore we did not include a "stop-list" in our method.

Next step was to implement the method in Dilemma. The graphical interface was in this stage much more mature and had a strong resemblance with the final one. When the new functionality had been added, we tried to increase the quality of the translation by calibrating the constants in the function (see section 4.4 for detailed information). This task was more difficult than we expected and will certainly require further studies. We finally found a configuration of values that did work satisfactory.

Third Step: Batch Production of Word Lists

Another sub-task in the project was to produce a list with every word in the source text together
with its corresponding translation. We thought that it would be of benefit to the user if this production could run as a background process and not interrupting the user from using the interactive part of Dilemma. This was a bit tricky, mainly because of the limitations inherent in Microsoft Windows. For example, we had to be very careful when assigning processor time to this process.

Many translations in the produced list were incorrect. One way to improve the results was to let Dilemma to estimate its own ability to translate a given word. We found a rather good method to estimate the quality of a translation (section 4.4). This estimation led to two (positive) properties of Dilemma. First, the list of translations contained fewer errors. Secondly, the end-user could, via the interface, tell Dilemma how good a translation must be in order to be presented.

4 Basic concepts

4.1 Working with Dilemma

The concept of the application is to support the translating process in order to make the translator work faster. Doing this, Dilemma first of all provides the translator with a way of finding previously translations of a word. This is a kind of browsing that is often done manually by the translator. With Dilemma this can be done in a number of automatic ways.

There are two ways of browsing in the interactive environment. The first is to examine the phrases that contains previous appearances of the word, the other is to ask Dilemma to translate the word.

4.2 String Alignment

As the string alignment problem is an essential part of our system we give a brief outline of it [Brodda 1976],[Gale et al 1991],[KVAL PM 1992].

Assume an alphabet \( A = \{ \varepsilon, a, b, c, ..., A, B, C, ... \} \), where \( \varepsilon \) is the zero-element, and two words \( x = \alpha_1, \alpha_2, ..., \alpha_m \) and \( y = \beta_1, \beta_2, ..., \beta_n \) from this alphabet, and distance measure \( D(\alpha, \beta) \) between pairs of letters \( \alpha \) and \( \beta \), we can define the distance between two words \( x \) and \( y \) as:

\[
D(x,y) = \sum_{i=1}^{m} D(\alpha_i, \beta_i)
\]

\( D \) must behave like any distance function (1-4). We also add some other characteristics to \( D \) (5-6). (x, y and z denote arbitrary words and "^\wedge" is the concatenation operation)

\[
\begin{align*}
D(x,x) &= 0 \quad & (1) \\
D(x,y) &\geq 0 \quad & (2) \\
D(x,y) &= D(y,x) \quad & (3) \\
D(x,z) &\leq D(x,y) + D(y,z) \quad & (4)
\end{align*}
\]
\[ D(x_1, x_2, y_1, y_2) \leq D(x_1, y_1) + D(x_2, y_2) \]  \hfill (5)

\[ D(x, e) = 0 \]  \hfill (6)

Relation four is usually referred to as "the triangle inequality". Relation five says that you may see more similarity between two words by considering the totality than you may see by considering smaller segments. Relation six is a special case and provides (together with especially (5)) us with an algorithm for finding the minimal distance between two words. \((x = x_m = \alpha_1, \alpha_2, \ldots, \alpha_m, y = y_n = \beta_1, \beta_2, \ldots, \beta_n)\)

\[ D(x_m, y_n) = \min(D(x_m, y_{m-1}) + D(e, \beta), D(x_{m-1}, y_{n-1}) + D(\alpha, \beta), D(x_{m-1}, y_n) + D(\alpha, e)) \]

This method is referred to as Elastic Matching [Brodda 1976]. It is not efficiently implementable but a good way to describe the concept. The problem of finding an alignment is of course the problem of finding the best path in the search space. In all authentic cases this is not a trivial problem.

The next step in order to make this result usable to us is to extend the aligning to work at another, higher, hierarchical level. In other words; we would like to match phrases-to-phrases instead of words-to-words. One way of doing this is described by Church and Gale [Gale & Church et al 1991]. We do not use their method but instead a method developed and implemented by Hans Karlgren. The model uses a distance measure that is based on differences in length of sentences. Pair of sentences are assigned to a probabilistic score that is based on the ratio of lengths of the two sentences and the variance of this ratio. This probabilistic score is used in order to find the maximum likelihood alignment of sentences.

4.3 Definition of elements

The translation process can be discussed in terms of set theory. A formal treatment is given in this section to enable an abstract discussion about how Dilemma works.

We will begin with some formal definitions. It is convenient to think of a corpus, \(X\), as a set of sets \(X_i\), where each \(X_i\) is a set of sequences, \(x_{ij}\), and the tags, i and j, are numbered to make the sequences correspond to words in a written text. The sets \(X_i\) correspond to a hierarchical level in the language. A hierarchical level is, in this context, referred to as some intuitive segment of text that consist of smaller parts of text segments at a lower hierarchical level (and the delimiters that separate these text segments). A corpus could be divided in the hierarchical levels document-section-paragraph-sentence-phrase-word-grapheme. When represented as a set of sets, the corpus only holds information of two hierarchical levels since this is enough for the following discussion. It is possible to convert any written text to this representation (i.e. if the delimiters to separate text segments at the hierarchical levels are known).
Example 4.3.1

The following text fragment is the first stanza of the Swedish national anthem. The set $X$ is the same text fragment represented as a set. $\text{delimiters}_\text{word}$ is the set of delimiters used to separate words and $\text{delimiters}_\text{phrase}$ is the set of delimiters used to separate phrases.

$$\text{delimiters}_\text{word} = \{\ "\}$$
$$\text{delimiters}_\text{phrase} = \{\ ",\",\ ";\"\}$$

"Thou old, thou free, thou mountain high North;
thou silent, thou beauty full of joy.
I greet thee, fairest land upon Earth,
thy sun, thy skies, thy fields of green."

$$X = \{(x_{11}, x_{12}), (x_{31}, x_{32}), (x_{13}, x_{33}, x_{34}),
(x_{41}, x_{42}), (x_{51}, x_{52}, x_{53}, x_{54}, x_{55}),
(x_{61}, x_{62}, x_{63}), (x_{71}, x_{72}, x_{73}, x_{74}),
(x_{81}, x_{82}), (x_{91}, x_{92}, x_{93}, x_{94})\}$$

This gives us a representation where a corpus is a ordered set of sets. Each member, $<X, i>$, is a 2-tuple where $X$ is a set and $i$ is a unique natural number (N). This makes each member unique. In short form a member is written $X_i$. The order over the set is $\leq$. $X_i \leq X_j$ iff $i \leq j$ and $i, j \in (N, \leq)$. In the same way each member of $X_i$ is ordered; $x_{ij} \leq x_{ik}$ iff $j \leq k$ and $j, k \in (N, \leq)$.

Corpus, $X$

$$X = \{X_i \mid X_i = \{x_{i1}, x_{i2}, ..., x_{in}\}\}, \text{ where:}$$

$$x_{ij} = \alpha_1, \alpha_2, ..., \alpha_m$$

$$\alpha_i \in A = \{a, b, ..., \"\, A, B, ..., \"\}$$
Equivalence relation, $\equiv$

\[
x_{ij} = \alpha_1, \alpha_2, \ldots, \alpha_n, \quad x_{kl} = \beta_1, \beta_2, \ldots, \beta_m \quad \alpha_i, \beta_i \in \mathcal{A} \\
x_{ij} \equiv x_{kl} \iff \alpha_i = \beta_i, \alpha_2 = \beta_2, \ldots, \alpha_m = \beta_m \iff \forall r : \alpha_i = \beta_i \\
( \Rightarrow n = m )
\]

Equivalence class, $X_m$

\[
X_m = \{ x_{ij} | x_{ij} \equiv x_{kl} \land x_{ij} \in X_j \land x_{kl} \in X_k \land x_i \in X \}
\]

Observe that $\equiv$ is simple string identity, and that $|X_m|$ is the number of occurrences of $x$ in $X$ that is equivalent to any $x_{ij}$ in $X_m$. This is modular with respect to preprocessing.

4.4 Finding occurrences of a word

As discussed, we want to provide the user with the possibility to view the source word $s$ in a context in the destination language. Later we will try to go one step further and use the aligned corpora to find words, i.e. $s_{ij}$ and $t_{kl}$, that are associated with each other.

In the special situation where we have two corpora which are translations of each other, this association can be very close to a translation. We call the two corpora source and target. The string alignment function is called $M$ and is designed to associate $S_i \in S$ to a $T_j \in T$;

\[
S = \text{source corpus}, \\
T = \text{target corpus}, \\
M \subseteq S \times T = \{ \langle S_i, T_j \rangle | S_i \in S \land T_j \in T \}, \\
M(S_i, T_j) \iff \langle S_i, T_j \rangle \in M
\]

Example 4.4.1

We use the same text fragment as in example 4.2.1 to illustrate $M$. The source language is Swedish and the target language is English.

\[
S = \{(\text{Du}, \text{gamlan}), (\text{du}, \text{fria}), (\text{du}, \text{fjällhöga}, \text{Nord}), (\text{tu}, \text{tyst}), (\text{du}, \text{glädjerika}, \text{sköna}), \\
(\text{Jag}, \text{hälsa}, \text{dig}), (\text{vänaste}, \text{land}, \text{uppa}, \text{Jord}), \\
(\text{din}, \text{sol}), (\text{din}, \text{himmel}), (\text{dina}, \text{ångder}, \text{gröna})\}
\]

\[
T = \{(\text{Thou}, \text{old}), (\text{thou}, \text{free}), (\text{thou}, \text{mountain}, \text{high}, \text{North}), \\
(\text{thou}, \text{silent}), (\text{thou}, \text{beauty}, \text{full}, \text{of}, \text{joy}), \\
(\text{I}, \text{greet}, \text{thee}), (\text{fairest}, \text{land}, \text{upon}, \text{Earth}), \\
(\text{thy}, \text{sun}), (\text{thy}, \text{skies}), (\text{thy}, \text{fields}, \text{of}, \text{green})\}
\]
\( M = \langle \{ \text{Du, gamla}, \langle \text{Thou, old} \rangle \}, \langle \text{du, fria}, \langle \text{thou, free} \rangle \rangle, \\
\langle \text{du, fjällhöga, Nord}, \langle \text{thou, mountain, high, North} \rangle \rangle, \\
\langle \text{du, tysta}, \langle \text{thou, silent} \rangle \rangle, \\
\langle \text{du, glädjerika, sköna}, \langle \text{thou, beauty, full, of, joy} \rangle \rangle, \\
\langle \text{Jag, hälsar, dig}, \langle \text{I, greet, thee} \rangle \rangle, \\
\langle \text{vänaste, land, uppå, Jord}, \langle \text{fairest, land, upon, Earth} \rangle \rangle, \\
\langle \text{din, sol}, \langle \text{thy, sun} \rangle \rangle, \langle \text{din, himmel}, \langle \text{thy, skies} \rangle \rangle, \\
\langle \text{dina, ångder, gröna}, \langle \text{thy, fields, of, green} \rangle \rangle \rangle \\

Illustration of alignment \( M \):

\[
\begin{align*}
\text{Thou old,} & \quad \text{Du gamla,} \\
\text{thou free,} & \quad \text{du fria,} \\
\text{thou mountain high North;} & \quad \text{du fjällhöga Nord;} \\
\text{thou silent,} & \quad \text{du tysta} \\
\text{thou beauty full of joy.} & \quad \text{du glädjerika sköna.} \\
\text{I greet the,} & \quad \text{Jag hälsar dig,} \\
\text{fairest land upon Earth.} & \quad \text{skönnaste land uppå Jord.} \\
\text{thy sun,} & \quad \text{in sol,} \\
\text{thy skies,} & \quad \text{din himmel,} \\
\text{thy fields of green.} & \quad \text{dina ångder gröna.}
\end{align*}
\]

We want to associate one representative member from each equivalence class in \( S \), \( i.e. \ s \in S \), since they all are equivalent in our terms, any member will do. The translation data set \( D_s \) of \( s \) is then:

\[
D_s = \{ \langle S_i, T_j \rangle \mid s \in S_i \land \langle S_i, T_j \rangle \in M \} \quad (\subseteq M)
\]

These phrases are exactly the phrases of interest to the user. They show a word from the source corpus in a context from the target corpus.

4.5 Translating words

For each translation candidate \( t \) to \( s \), there also exists a translation data set, \( D_t \). This set of course common to all members in the equivalence class of \( t \):

\[
D_t = \{ \langle S_i, T_j \rangle \mid t \in T_j \land t \equiv \forall \land \langle S_i, T_j \rangle \in D_s \} \quad (\subseteq D_s)
\]

The problem of finding the translation candidate in \( D_s \) that best embodies our association can be regarded as the problem of finding a function that gives that candidate the highest value. We call this function \( G_{(6,9)} \) and the value is called the translation grade.

Depending on what association we want to find we have to design a function that evaluates to the highest value for the pair that best represents our association. We try to find a function that represents a translation, we have to find properties that are valid for a translation. The following function is built up by four terms that all are qualities that a possible translation should have.

The first term (scaled by \( k_1 \)) is based on the assumption that, if there are many occurrences
of the word $t$ in $D_i$ then $t$ is likely to be the translation of $s$. The pair gets $k_1$ points for each existence in the target corpus.

The second term (scaled by $k_2$) says that $t$ is more likely to be a translation of $s$ if $t$ occurs at the same position in $T_i$ as $s$ in $S_j$ and $<S_j, T_i> \in M$. This part looks a bit odd because we calculate the mean value of the distance between $s$ and $t$. We also have to scale $D_i$ with $k_2$ to make it comparable to the other terms of the function in a form of normalisation. Of course both these are cut off to make the computation less expensive.

The third term (scaled by $k_3$) says that $t$ is rewarded if $t$ occurs the same number of times as $s$ in the whole corpus (i.e. $S$ and $T$).

Finally, the fourth term examines the correlation between the lengths of $s$ and $t$. This term is also scaled by the same reason as the second term.

$$G_{(s,t)} = k_1 \cdot |D_i| - k_2 \cdot \left( \sum_{j=0}^{\infty} \frac{|D_i|}{|T_i|} \cdot (|s_j| + |T_i|) \cdot \frac{1}{|S_j|} \right) \cdot |D_i| - k_3 \cdot |S_m| \cdot \frac{1}{|T_m|} - k_4 \cdot |t| - |s| - |D_i|$$

where: $|s_j| = i'$, (position of word $s_j$ in phrase $S_j$)

$|t| = n \Leftrightarrow t = \alpha_1, \alpha_2, ..., \alpha_m$

$|s| = m \Leftrightarrow s = \alpha_1, \alpha_2, ..., \alpha_m$

$$G(s) = t \Leftrightarrow <s,t> = \max G_{(s,t)}$$

All though this function will work in some cases "without" constants (e.g. with all constants assigned to one), we can improve the result by finding the optimal configuration of constants. The problem is that there exists an optimal configuration for each bitext. As an example, consider two phrases in languages which has a low correspondence between positions of words. In this case $k_2$ should be small. A bigger problem is that constants differ within the same pair of languages, because of the ways different writers (and translators) use the language.

This leads us to the conclusion that we can not hope to find constants that will work optimally on arbitrary corpora.

**Finding a configuration of constants**

One way to find a good (but not optimal) configuration of constants is by calculating their values for a representative set of words and corpora. Starting with a small set of words it is easy to find a working configuration (of constants). By increasing the number of words in the set, and adjusting the values of the constants to represent the desired association, their values improves. The reason for this is that the range of each constant gets smaller. It is obvious that the quality of the result depends completely on the chosen set of words. We have to choose words carefully
to find constants that will work well on other corpora.

It is necessary, and easier, to use words from both corpora to make the association good in both directions (i.e. if that is how the association is going to be used). It is easy, but erroneous, to think that this is not necessary because of the symmetry of the function. The fact is that the function is not necessarily symmetric when it is utilized to compute translation grades, or in other words, the pair with maximum translation grades may be different in the two directions. The reason for this is that the properties of the translation data set varies between different directions, another is that words often have synonyms and several meanings.

This can be expressed in an abstract way as: given a source text \( S \) and a target text \( T \) and we have the functions \( G_{s,t} \) and \( G_{t,s} \) with their natural interpretations (see section 4.5). Then if \( s,t \in S \) and \( G_{s,t}(s) = w \in T \) and \( G_{t,s}(w) = t \), it is not generally true that \( s = t \). As an example, consider a situation where \( s \) is a synonym to \( t \). Then they both are potential translations to a \( w \in T \). It is possible that \( G_{s,t}(s,w) \) may be the highest translation grade for \( s \), even though \( G_{t,s}(w,t) \) is the highest for \( w \). As an example, if \( S \) is in Swedish and \( T \) is in Finnish, the words could be \( s = "gosse", \ t = "pojke", \) and \( w = "poika".\)

**Properties of a correct translation**

When finding associated words we want to decide how good an association is. We would like to measure the quality of an association. This measure can be used in a number of ways, but the most obvious way is naturally during producing lists of correctly associated words, e.g. if we can find a threshold that separates the correct associations form the erroneous.

Fortunately the translation grades function \( G \) is well suited for computing such a measure. Recall that the function is a sum of four terms, each representing a desired property. As the function is formulated only the first term \( (k_1 \cdot |D_t|) \) adds to the values while all the other subtract. This can be reformulated: a perfect association should have a big value in the first term and zero values subtracted in the other terms.

Or informally: *The perfect association of a word in the source corpus is the word in the target corpus that occurs in all phrases of the translation data set \( D_t \), always in the same position as the source word, with the same number of occurrences in the corpus and the same length as the source word.*

The most useful way to use this result is to calculate the ratio of \( G / (k_1 \cdot |D_t|) \) since this value is not dependent of the size of the translation data set \( D_t \). The ratio is 1 if the association is perfect and less if it is not. Observe that the ratio may be less than 0 in some cases. It depends completely on the configuration of constants that is used. This is not be a problem if the constants are chosen suitably.

Now we have a measure that is suited to evaluate the quality of an association. We can set a threshold, which depends on the constant chosen, and consider associations with a quality measure higher than this threshold good enough for a special purpose (i.e. take part in a list of correctly associated words).
4.6 Morphosyntactic analysis

As the total size of the corpora (the material) influences the quality of the translation, it is tempting to believe that the supreme way to improve the ability of the applications to translate is to enlarge the material. This is wrong for at least two reasons.

First, when enlarging the material we also enlarges the number of different words\(^1\). Surely this does indeed improve the quality when translating words that are frequent in the material. The problem is that as the material grows it also introduces words that are difficult to translate and which do not get easier to translate.

The other reason is that this is not possible in the computational environment we are working in. The corpora must be analyzed interactively, i.e. while the user is waiting. To be sure that the words occur sufficient numbers of times we have to use a extremely large material.

We have found two ways of improving the quality of the translation in an interactive environment. The first, and most natural way, is to make it easy to select among the corpora. The assumption is that this makes the interesting words occur more frequently, which makes it easier to find the right translation. We let the user arrange the corpora in subjects and specific subjects.

So far, the translation depends solely on comparing by statistics. The other way of improving the translation quality analyses the corpora by using linguistic analysis methods. If we could find a way to make the equivalence classes bigger we would get a bigger translation data set without enlarging the material. This is done by modifying the equivalence relation. In order to do so we use a morphosyntactic analysis. The effect of doing this is that the number of equivalence classes reduces while both \(D_s\) and \(D_t\) get larger, by using linguistic analysis methods. Since we only try to translate words that are representants of an equivalences class, we conclude that these get fewer and their translation data set grows. Additionally, the extra information (e.g. part of speech, inflectional form, etc) provided by the analysis can be utilized in the function.

5 Developing the application - stages

We have earlier in section four defined the problem of finding text fragments and translations in terms of defined mathematical elements and elementary set theory. We will now describe how we concretized the abstractly defined elements in terms of objects and how these objects solve the problems by interactions.

We will in this section present the identified objects, how they are organized in Dilemma and how they achieve the objectives of the project. The section is divided into two parts; first we explain how Dilemma solves the problem of presenting previous translations and then how Dilemma solves the problem of translating a given word.

Generally, we didn't have much trouble in identifying the objects. Certain objects were easy to identify because they had natural connections to previously defined mathematical elements.

---

\(^1\) Number of equivalence classes.
Others provide services that are not relevant when describing how Dilemma solves the tasks and are not included in the description below.

5.1 Interactive tool with hierarchical levels

*Representation*

We view the system as a collection of interacting objects. The primary task for Dilemma is to look up previous text fragments where a given string occurs and present its corresponding translation to the user. The following key objects have been identified to achieve this task (see figure 5.1.1):

- *Corpora* is the supervisor and offers a interface between the presentation and the functionality. It contains two Documents (will be called S and T in the following text, denoting source and target Document respectively) and an alignment between these, called Meta. This object has no corresponding mathematical element in section four, but is nevertheless important when describing how Dilemma solves this task.

- *Meta* represents the string alignment function, called M in section 4.3.

- *Document* is a document containing text and corresponds to a corpus in section 4.2.

- *CDilemmaDlg* is a dialog shown to the user and works as the graphical and communicational interface between Dilemma and the user.

We will now describe how this system solves the first task; namely to find previous translated fragments of texts. This description is deliberately abstracted from the discussion about the actual calls to the methods offered by the objects. We believe that it would only add extraneous detailed information that can be found in the source code. For more detailed information, we refer to the source code.

When the user asks Dilemma to look up translated text fragments, given a string in the source text (S), CDilemmaDlg uses Corpora to collect the elements (∈ M) in Meta where the position of the string is contained within the source part (∈ S). Corpora will only collect elements down to a specific hierarchical level, which is specified by the user. For example, a typical level could correspond to paragraphs in the text.

When the elements have been collected, CDilemmaDlg asks Corpora to look up the actual text fragments (∈ T) in the target Document (T). Corpora uses the target Document (T) to look up the text fragments described by the elements collected. Finally, CDilemmaDlg presents these to the user.
Presentation

The design of the Dilemma interface is discussed as a part of this section. The discussion concerning interface issues is organized as follows: Section 5.1 discusses the design of the first interactive interface; Section 5.2 discusses the word translation including, configuration dialog and status bar; Section 5.3 discusses batch analysis, main menu and the subject dialog.

The design of the Dilemma interface is based on some aspects that are important or will be in the future. As these aspects highly affects the look of Dilemma we now give a short discussion about them. In short these are the following:

- The application will be used together with WordPerfect® for Windows. This means that the application should have a similar appearance as WordPerfect itself and other third part applications. In MS Windows terms this means essentially that Dilemma should be a dialog box instead of a frame window.

- The freedom of designing the interface is limited by Microsoft Design Guidelines [Microsoft Press 1993]. These guidelines highly constrain the freedom of designing interface in order to make it similar to other applications. This should result in simplified usage for the application user.

- The design is limited to the routines that are available in Microsoft Visual C++ which was the only tool used during the development of the application.

Having regard to these restrictions we designed an interface that appeared a bit plain in the first phase (figure 3.4.1). The main window is a dialog-box containing an edit-box where the user
enters a string, a list box to show phrases and a multi-line edit-box to show a paragraph. Figure 5.1.2 shows the complete window as it looks in a later version\(^2\). The figure is provided to readers that is not familiar with the terms of MS Windows.

Besides guidance and restrictions we tried to keep the size of the Dilemma main window as small as possible. We believe that this is important to make the application flexible and useful together with WordPerfect\(^6\) for Windows, as both applications will appear at the screen simultaneously.

The window also has a menu and a status bar. The menu contains selections that the user do not have to use frequently. The status bar displays information that is useful to the user but not necessary when searching phrases. This is why the status bar can be turned on or off\(^3\). The status bar is discussed in detail in section 5.3.

Another key point is that it should be possible to do all actions in different ways. This means that user should be provided with different ways to give the commands. This is important because of the differences in work habit between novice and skilled users. Skilled users probably prefer to work with short-cuts (such as key-combinations) in order to make their work faster, while there should be a simpler way (such as menu selections) for novice users. Another reason for enabling both mouse and keyboard interaction is that there still may be some users using MS Windows whiteout mouse (we

\(^2\) In this version word translation and batch analyses has been added.

\(^3\) See guideline no 9.
disregard the fact that some users may want to use a pen-device).

From the first test version of Dilemma, we have provided a way to get information about the program by displaying an about dialog box. This is supplied by all MS Windows applications (and is recommended by the Design guideline). The dialog is shown in figure 5.1.3.

Init-files is another feature that is supplied by several MS Windows applications. Dilemma uses its ini-file to store information about configurations and other setups as they were set the last time the application was exited (e.g. position on the screen and selected corpora). An example of a inst-file is bundled with this document (see Appendix B).

In this phase we also had to decide how to implement the interface. This includes identifying proper objects and their methods. Since this is not an interesting part of the interface design, especially not in Microsoft Visual C++, we only give a summary of the objects and their interaction of services (figure 5.1.4).

![Diagram of interface objects and their interaction](image)

**Figure 5.1.4 The interface objects and their interaction.**

**Using the application**

When using the application, to find a phrase, the user enters a phrase (i.e. any string of characters) into the edit-box, then presses the return-key or chooses fras (i.e. phrase) in the sök-menu (i.e. find menu). This instructs the application to find a pre-specified number of occurrences of the phrase in the source corpus. The number of occurrences is specified in a configuration dialog box (see 5.3). The result is presented in the list-box. Each phrase in the list-box can be displayed at an higher hierarchical level by pressing the phrase (using the cursor or the tab- and/or arrow-keys). When this is done the paragraph (i.e. higher hierarchical level) containing the phrase is displayed in the multi-line edit-box. In addition it is possible to ask Dilemma to show the aligned paragraph (i.e. in the corpora). This could be done by pressing return-key after selecting the phrase, or by selecting stycke in the sök-menu, or by clicking the status bar, or by double clicking the phrase.
5.2 Translating words

**Representation**

The second main objective in the project is to find a translation of a given word. Now, suppose that Corpora only collects elements in Meta where the given word is contained within the source part. This set of elements corresponds exactly to the translation data set $D$, defined in section four. Corpora then incorporates all the information needed to be a basis for realizing the translation function $G_{(s)}$. Consequently, the new identified object, called Translation, will use Corpora when to find a faithful translation. We will now show a more complete picture of the most essential objects in the system:

![Diagram](image)

*Figure 5.2.1 The translation process.*

The procedure of finding a translation is, in the first phase, identical to the procedure of looking up previous translations. First, CDilemmaDlg asks Corpora to collect all elements in Meta where the position of the word is contained within the source part. Then Translation uses the information collected by Corpora to find a correct translation. The method used in Translation for this task is described in section 4.3 (for comments, see section 6.2). CDilemmaDlg asks Translation for the three best translation candidates and then presents them to the user.

**Presentation**

In this development phase the interaction design first seems to be a small extension to the existing window. However, this is not the case. When the application functionality extends to deal with translation of words, the interface has to provide the user with more elaborate control mechanisms in order to make these features easy to use. This depends first of all on the number of configuration choices that the translation of words introduce. It also depends on the enlarged number of information that the user has to consider.

In order to present suggested translations to the requested word, the main window was equipped with a combo-box (figure 5.1.2). The combo-box shows the three words that evaluate to highest translation grades. It is possible to scroll the combo-box to get two more words.

In order to tell Dilemma to always translate words, the user has to consult the configuration
dialog (figure 5.2.2). By enabling the word translation, the user instructs the application to translate word-to-word if possible (i.e. if the phrase does not contain delimiters). Besides this choice the configuration dialog provides the possibility to select the number of phrases to find (and show in the main window). This number directly adjust the maximum size of the translation data set. The reason for letting the user decide this size is that it is a trade-off between quality and speed. This will probably be popular among user with old and slow computers.

Other configurations are the direction of bi-texts (i.e. from source-to-target or target-to-source). When this option is selected the search will take place in the target corpora and phrases (and sections) will be presented from the source corpora. It is also possible to tell the Dilemma to copy the contents of the clipboard when the application is started, and (or) copy form the application to the clipboard when the application in halted.

As it is a bit inconvenient to examine the configuration dialog to check which selections are made, the most important selections are also shown in a status bar (figure 5.2.3). The status bar also shows some other information that we believe can be important to the user. These are 1) source hit percent that shows the number of occurrences of the seek phrase in the source corpus, in relation to the requested number of source hits. 2) The translation value that gives the user an idea of the quality of the translation that is suggested at the first position of the combo-box (see 4.4 section Properties of a correct translation for details).

In addition the status bar also implements a convenient way to change some of the configurations. This is done by clicking the status field that shows the selection. For instance the bi-text direction may be changed in this way. Of course the selection also takes place in the configuration dialog.

The development of a interface, for word translation, also involved some major modification and extensions to the application menu. The entire menu is discussed in 5.3.
5.3 Batch analyses

Representation

A natural extension of translating one word is to ask Dilemma to translate several words in the source text and present them in a separate dictionary. Consequently, we decided to design and implement a process that produces a word list in the source text together with their corresponding translations.

In order to not disturb the user during the translation, we implemented the process as a background process. The background process will only be able to execute when the processor is idle. The user can, at any time, stop the process to receive the list produced so far.

The list produced by the background process has two columns; the first column contains words from the source text and the second contains the corresponding translations. A typical list can have the following appearance:

Example 5.3.1

| attainment | genomförande |
| bankruptcy | förstått |
| banks      | utfärdade   |
| base       | basmaterial |
| basis      | föreliggende |
| batteries  | batterier   |
| belgium    | belgien     |
| better     | bättre      |

In section 4.5 we presented the properties of a perfect translation. A perfect translation has a corresponding translation grade that can be calculated a priori (by the statistical method described in section 4.4). This observation was shown to be useful for Dilemma. We let Dilemma to estimate its own translation ability by comparing the translation grade calculated for the suggested translation with the optimal one.

One effect of this feature is that the user can set the threshold for when to accept a translation as valid. Consequently, the number of translation errors can be reduced in the produced list by only including valid translations. In example 5.3.1, the user has set a threshold that will only include a translation if the calculated translation grade is not less than 90% of the optimal.
Presentation

The majority of the interface is not affected by the batch analyses which is the object of this phase. The extension was restricted to menu and status bar modifications. In the status bar the batch percent was added. This field displays the progress of the batch analysis.

In the menu the selection ordlista (i.e. word list) was added. This selection opens a sub menu which consists of starta (i.e. start) and avbryt (i.e. cancel). Since the rest of the menu only has been discussed briefly in the previous sections, we now give a short presentation of the entire menu.

The Arkiv menu (i.e. file menu) contains selections, configuration, subject matter, constants (this selection is only shown in a evaluation version) and exit. The Redigera menu (i.e. edit menu) contains selections for enable/disable the status bar and selections to edit the text in the application's main window. The edit selections are not implemented as menu selection but are provided as reminders to the short-keys. In the Sök - menu (i.e. find menu) the user can instruct the application to find occurrences of a phrase, a word translation, the opposite part of the shown paragraph (if any) and instruct the application to start or cancel the batch analysis.

Another feature that is added to the interface at this development phase is the subject matter dialog. The subject dialog enables the user to select among previously registered subjects. The subjects are ordered in an hierarchical way in order to make it easy to find specific subjects even if the number of subjects are big. Figure 5.3.2 shows a situation where a user selects the subject Etablering inom EG which is a part of the main subject EG. If the user select Huvudmeny (i.e. main menu), the dialog will jump up one level and show the specific subjects of Huvudmeny. How the subjects are ordered has nothing to do with the underlying file system. On the contrary, the main idea here is to conceal the underlying file
system while working with the application. To users that are interested in file information there is also the possibility to select the button *Filer* (i.e. files). This makes the dialog window bigger in order to give space to show the files of the subject. These are labelled: source file; target file; meta file and batch file. Information about subjects and there files are stored in the ini-file.

6 Implementation Issues

Our objective was to create a prototype for evaluation. Therefore, we have not discussed efficiency issues in the same extent as if we would if we were about to implement a product. In this section we will talk briefly about some implementational aspects concerning the identified objects in the previous section. We will also mention some aspects concerning effectiveness and pointing out where the potential bottlenecks are located. We will avoid implementational details, as they can be found in the source code.

6.1 Interactive tool with hierarchical levels

*The class Meta*

We are assuming that the string-alignment module Para produces a file which contains a tree where each node denotes a binding between two text fragments; one from the source text and the other from the target text. Each node in the tree corresponds to an element in \( M \) and the whole tree is a relational variant of that function (see Appendix C). That is, every node is corresponding to an alignment between two text fragments at a specific hierarchical level. The depth of the tree corresponds to the number of hierarchical levels used in Para during the aligning process.

Each node in the external file has a label. This label denotes the hierarchical level of that node. The length of the label is currently one (an ANSI character) and the depth of the tree can therefore not exceed 255. We do not believe that this limitation is critical, by arguing that there probably is not that many natural hierarchical levels in an arbitrary text.

The class Meta has an internal representation of the file produced by Para, which is built up during the initialization. This internal tree allows Meta to use binary search, \( O(\log N) \), when looking for elements.

*The class Document*

The source text and the target text are read into the internal memory from external files. The size of the texts is limited by available internal memory. We have been working with bi-texts with sizes up to 100 kByte; far from the memory limit of a conventional computer system today. In order to handle larger files, the class can easily be rewritten to use secondary units directly instead of the internal memory.

A class called Document is used to represent the read files (see section 5.1). The class Document uses a naive, brute-force, algorithm when searching for a string in a text. This
algorithm is locally defined in a method and can be easily replaced by a more efficient (and thereby more complex) algorithm. In order to be able to concentrate on the functionality of the prototype, we decided to defer the implementation of a better algorithm [Boyer et al 1977]. We have not found that this decision affected our ability to evaluate the usefulness of the prototype.

The class Corpora

One essential task for Corpora is to collect sets of elements from Meta, given a search string (or a word) from the source language. Each set is a hierarchical path which describes, at every hierarchical level, where to find a text fragment in the source document including the search string and where to find its corresponding text fragment in the target document. This set is represented by a class called Hit.

Example 6.1.1

Suppose that a search string x is found at position 42. If the alignment has three hierarchical levels, the three elements in Meta constructing the hierarchical path could look as follows (see Appendix C for more information):

\[
\begin{align*}
  d,0,100,0,110, & - \text{hierarchical level 0 (document)} \\
  s,0,60,0,65, & - \text{hierarchical level 1 (sentence)} \\
  p,40,50,43,55, & - \text{hierarchical level 2 (phrase)}
\end{align*}
\]

These elements are put into an object of type Hit.

Corpora incorporates a bag. This bag contains elements of the type Hit. The elements in the bag is ordered with respect to the appearance of the search string in the document. Because of the structure of the bag, insertion and extraction is done in constant time.

When all elements are collected, Corpora can extract the text fragments that corresponds to the elements in the bag. The extraction is immediate, as the documents are internally stored in arrays.

6.2 Translating words

The class Translation

By far the most expensive operation in Dilemma is to find a adequate translation. The reason is that the method defined in section four is based on statistical information and that requires considerably computational effort. The information used as the basis for this calculation lies in Corpora (see section 5.2) and is passed as an argument during the initialization of Translation.

We have not implemented the method exactly as defined in section four. The reason is that it would require too much computing and that would hinder the user from accessing the interactive part of Dilemma. We have done the following restrictions:
The third term of the function $G_{(a,b)}$ in section 4.5 is calculated only for a limited set of words in the source text that seems to be promising translations. That is, first the object Translation calculates a translation grade for every word in the target text occurring in Corpora, but considers only the properties corresponding to part one and two in $G_{(a,b)}$. Then the top five (or less) words is extracted. Finally, Translation adds the points corresponding to the third part of the function $G_{(a,b)}$ for each extracted word.

The fourth term of the function $G_{(a,b)}$ has not been included at all. We did not found that this property increased the translation quality significantly and was therefore excluded.

The class Translation uses a table, named WordTable, for storing the words and their translation grades during the analysis. Every word in the table must be unique and can be used as a key in a hash table. Not surprisingly, WordTable is a hash table and uses linear probing for insertion and extraction of words.

The algorithm for generating a key is based on the individual characters in the word. Again, the algorithm is far from optimal, but suits our needs during the evaluation of the prototype.

6.3 Batch analyses

One problem after the implementation of the background process, was to make it run smoothly among other active applications in the Windows' environment. If we gave the process too much priority, other applications would be suspended. On the other hand, if we gave it to little process time, it would not finish its task within an acceptable time. Our solution was to let the background process, in run-time, count the numbers of active applications and then set its own priority. Generally, it is up to the applications in Windows to let go of the processor. Of course, in a true time-sharing environment (which is not the case for Windows), it is the obligation of the operating system to share processor time among the applications.

The background process has a separate Corpora-object and therefore the user can change subjects during the background translations. As the statistical method for finding a translation is not depending on the languages, the direction of the translation is specified by the user. This specification is implicit; the direction of the translation will be the same as the current direction in the interactive part of Dilemma.

When a Translation-object has calculated the translation grade for a translation, given a word from the source language, the quality of the translation can be estimated in constant time. Recall the properties of a correct translation. For a correct translation, it is only the first term in $G_{(a,b)}$ that can have a value not equal to zero. Consequently, we can get a measure of the translation quality dividing the translation grade with the result of multiply the constant $k_1$ with the size of the bag in the Corpora-object (which corresponds exactly to the first term $k_1 \cdot |D_i| \in G_{(a,b)}$). Both $k_1$ and the size is given, so the measure can be done by a simple division. The closer the result is to one, the better is the quality of the translation.
7 Evaluation

Since the work is divided into two parts - building a prototype and investigating methods to aid manual translation - the evaluation is also divided into two corresponding parts.

Methods

Evaluating the methods can be done easily by measuring some characteristics. Questions that we found interesting where:

- How often does Dilemma present a correct translation?
- What is the quality of the generated lists of words?
- How well does the application's estimation of the translation quality correlate with a translator's opinion?

Since the quality of the generated list of translations depends on the threshold used and the quality of the bi-text alignment, we can only give a typical example. Figure 7.1 and 7.2 shows how the translation quality and size depends on the threshold.

![Figure 7.1](image) Number of correct translations in relation to the threshold. Size of source corpus: 1502 words, 243 different words.
Application

When evaluating prototypes one always finds plenty of room for improvements in the graphical interface and other implementation details; that is one of the objectives when implementing a prototype. When developing prototypes it is important to evaluate several times, during the development of the prototype. We have used experienced translators and computer users to validate our ideas about the application. These users had their own prototype copies for longer periods of time to be able to make fair judgements about the prototype.

Since these test persons have been the same during the entirely project, and their opinion have guided the design, we did not find it meaningful to ask them about there opinion about the final prototype. We had to use two other test persons. The questions that we want to find answers to is the following:

- Is the graphical interface of the application easy to understand and use?
- Do translators find Dilemma useful?
- How would a translator use the application?

This time the test persons were an experienced computer user and a translator. In the experiment the users were asked to translate a short abbreviated version, of a legal text from an existing bi-text. The idea was to let the users use Dilemma and the original bi-text to translate the text. Here are some results and comments from this test:
Users are sure that they will learn and understand the graphical interface. They are able to work with the application after a short introduction.

- Users will rather operate the application using keys, than a mouse.
- One user didn't like the appearance of the list-box (We redesigned the list-box.)
- Translators are more interested in phrase browsing then word translations.
- Translators find it useful to look up words, even if they know how the meaning of the word. This look up is done to see how the word is used in a special context.
- Translators think that they would use Dilemma in there work.

8 Discussion

Dilemma has shown to be useful for manual translators according to our qualitative tests. Still, Dilemma is a prototype and requires further study and improvements in order to make the prototype a natural integrated part of all translation phases described in section 2.3.

We will now point out some of the improvements and further work that would increase the usefulness of Dilemma.

Algorithms

We have in this report emphasized the fact that Dilemma is a prototype and not a running commercial product. The prototype has been effective enough (with respect to speed) for evaluation. Some algorithms used in Dilemma are, however, not optimal. There are at least two implementational details that can be improved. First, the string searching algorithm currently implemented in Dilemma is naive and inefficient. There exist many well-documented fast string searching algorithms that would improve the string searching process [Boyer et al 1977]. Secondly, the management of the words that are considered during the translation process (class WordTable) can be improved further (for example, better implementation of hash tables and improved algorithms for key generation).

Functionality

One extension of Dilemma's functionality is to present translations of text fragments instead of words. Translating words is a special case (the lowest hierarchical level). The method for presenting text fragments at higher hierarchical levels should not be a difficult extension. Probably the method defined in section 4.5 will work satisfactorily at any hierarchical level, but we have not verified this assumption.

Another extension in the interface is to let the user navigate among the different hierarchical levels in the text. Dilemma supports several hierarchical levels, but the user can only switch between two levels (for example between phrases and paragraphs). The problem of navigating in large volumes of information is out of the scope of this report and is therefore deferred to future
work.

**Preparatory Work**

Dilemma must have previously translated text, preferably containing similar information as the one being translated, in order to be useful for a translator. Essentially it comes to the problem of material acquisition. The most suitable material is probably the most recently translated texts a translator has produced. The process of collecting, constructing bi-texts, and categorize sets of bi-texts is a crucial factor in order to make Dilemma a powerful tool for manual translation. How to make this process effective as possible is a problem that requires further studies.

Dilemma does not support an interface for including new subjects. Instead the user has to manually edit the ini-file associated with Dilemma in order to include new subjects and specific subjects. This must be solved to facilitate the preparatory work.

**Methods**

We found that the heuristic methods of finding associations between words could be improved by adjusting the constants. We determined these constants by a simple twiddling algorithm. A preferable way would be to find a way to generate a configuration of constants by analysis of the bi-text that is currently used as a basis for translation. We think that this is possible and that the answer may be found in the area of machine learning or even in mathematical statistics.
Appendix A

Design Guidelines by Microsoft

We have tried to follow the design guidelines stated in "An application design guide", published by Microsoft Press [Microsoft Press 1993]. Here follow the guidelines relevant to the graphical interface provided by Dilemma.

In general, the author argues that the best interface is often the one that is hardly noticed. This is clearly true, but can easily be forgotten when designing the interface. The interface should be an efficient one prior to a pleasing appearance. A combination of these two attributes is ideal and will probably require iterative changes in the design. There are at least three important principles that a well-designed application should follow:

- **User Control**
  The user should always be in control of the application, not vice versa. Therefore, the application should be as interactive as possible and avoid restrictions in the interactions.

- **Directness**
  The users should be able to accomplish their tasks in a direct and intuitive way. Direct manipulation of objects is often easier than using commands.

- **Consistency**
  The application should be consistent in at least two ways; consistent with the real world and consistent within and among applications. First, the mapping between objects in the application and the real world should be intuitive. Consistence between applications helps the user to understand the application in less time.

During the project we have had these principles in mind in order to achieve a good interface. Unfortunately, we were forced to do some compromises, as the time was limited. Additionally, there where more specific guidelines relevant to the interface of Dilemma:

1. If an object is so small or thin that pointing or clicking to select it would require extremely precise mouse positioning, provide a "hot zone" around the object to increase the area where clicking will select the object.

2. Assign single keys for frequently performed, small-scale tasks.

3. Function key should not be the first choice for shortcuts; they are easier to localize but may be harder to remember because they are mapped to functions arbitrarily.

4. Application may allow the user to press ESC to cancel or interrupt lengthy operations.
5. When a window is reactivated, the focus and the selection should be displayed in the same locations as when the window was last active.

6. When possible, progress indicators should include a way of pausing and resuming lengthy operations.

7. Application windows should always include a title bar that contains at least the title of the application, a Control-menu box and a Minimize and Restore (or Maximize) button.

8. It is suggested that windows that always exist on top of others be given a grey shadow to visually distinguish them from other windows. There should be an entry "Always on top" in the windows's Control menu.

9. Because the status bar takes up space that could be used to display data, applications should always provide a way for the user to control whether the status bar is displayed.

10. Application may allow the user to press ESC to cancel or interrupt lengthy operations.

11. Applications may implement control bars to provide quick and convenient access to frequently used choices and commands.

12. In general, cascading menus should be avoided whenever possible.

13. Whenever possible, an menu should contain no more than four groups of items, with three or four items in each group.

14. The names of dialog box commands should be followed by an ellipsis ("... ").

15. Control menus for dialog windows contain only Move and Close commands.

16. The File menu frequently includes optional items such as the most recently used (MRU) list of files.

17. The MRU (list of the Most Recently Used items) should not be less than three or more than eight items. Each item is placed next to a number which can be used as a mnemonic access character. The most recently used item should be at the top of the MRU.

18. The Help menu should always be the rightmost menu of the menu bar.

19. The Help menu should contain the items Contents, Search for Help On, How to Use Help and About in the order listed.
20. The View menu includes commands for changing the user's view of the data in the window. The View menu may also include commands for controlling the display of interface elements, such as status bars and control bars that are a fixed part of the application.

21. Three-dimensional buttons that set properties should be shown in the depressed position whenever the specified property is in effect.

22. Graphical labels on buttons are useful in reducing clutter when many buttons are presented in a small space.

23. Unfold buttons (such as Options >>) can be contained in a dialog box, which expands the dialog to include additional options.

24. In a standard list, a horizontal scroll bar may be added at the bottom of a drop-down list if a particular item is too wide for the list.

25. A value typed into the text field of a spin box should be validated either immediately or as soon as the user navigates away from the spin box.

26. The first and the last word of all labels should be capitalised. In addition, capitalize all initial letters of all other words in a label, except for articles, coordinate conjunctions, prepositions and the to in infinitives.

27. Provide a unique mnemonic access character for labels of controls to which the user needs direct keyboard access.

28. A dialog box that is not movable should not have a title bar.

29. After the dialog is expanded, applications may optionally leave the button active and the chevrons facing the opposite direction (for example, "Options <<").

30. Every message dialog has a title bar. Each message dialog should include a graphical symbol that indicates what kind of message is being presented. The three types of messages are information, warning and critical messages.

31. Message boxes should contain only command buttons control.

32. The most appropriate places for buttons in a dialog box are at the right or at the bottom, where the buttons will be seen after the user has already scanned the relevant information. The first arrangement should be given priority.

33. The most important button should be placed at the top.
34. Help buttons in message dialogs are optional but highly recommended, especially for warnings and critical messages dialogs.

35. Some elements which are recommended in a About dialog in the order listed:
   - Official name of the application
   - Version number
   - Copyright statement
   - An icon associated with the application to the left of these lines
   - System information (optional, but highly recommended)

36. Steps for loading the application:
   1. Display the application window.
   2. Display the startup message (which can contain the same information as the About box).

37. Once the application code is executed, it becomes the application's responsibility to inform the user if insufficient memory is available for performing specific operations.

38. When the user quits an application, the current customizations should be saved so that they can be set up in the same way the next time the user invokes the application.

39. To complete successfully in international markets, applications should ensure that their interfaces can be easily adapted to accommodate differences in languages.

40. Interface text should be stored as resources file rather than being included in the source code for the application.
Appendix B

An example of Dilemma’s ini-file

[Main]
;;; This part contains options that are automatically saved after each session.
;;; maxHits=8
translateWord=1
defaultDirection=0
copyStart=1
copyExit=0
subject=EtabloringinomEG
xPos=526
yPos=153
statusBar=1
; Constants that modifies the translation grades
constant1=2.
constant2=1.
constant3=0.5
constant4=0.0
; Not used
;;; This part contains options that one might want to save after a session.
;;; These might be implemented in the future.
;;; Hierachial level to use as translation data set.
level=2

;;; This is the subjects and specific subjects that are shown in Dilemma.
;;; 'Huvudmeny' is at the top level - the root.

[Huvudmeny]
priorSubject=
specificSubjects=/EG/ /Kodning/ /Diverse/
name=/Huvudmeny/
description=Huvudmeny

[Diverse]
priorSubject=/Huvudmeny/
specificSubjects=Flygplats RobertN
name=/Diverse/
description=Diverse okategoriserat material

[Kodning]
priorSubject=/Huvudmeny/
specificSubjects=DilemDlg DilemSubj DilemInst
name=/Kodning/
description=Dilemmas program-filer

[DilemDlg]
priorSubject=/Kodning/
specificSubjects=
name=Dilemma Dialogen
description=CDilemmaDlg är klassen för Dilemmas huvudfönster. CDilemma-Dlg ärver från MFC-klassen CDdialog.
language=Engelska till Svenska
source=c:\usr\corpora\dilemdlg.eng
destination=c:\usr\corpora\dilemdlg.swe
meta=c:\usr\corpora\dilemdlg.met
batch=c:\usr\corpora\dilemdlg.bch

[DilemSubj]
priorSubject=/Kodning/
specificSubjects=
name=Amnesialogen
description=CDilemSubjDlg är klassen för Dilemmas ämnesdialog. CDilemSubjDlg ärver från MFC-klassen CDdialog.
language=Engelska till Svenska
source=c:\usr\corpora\dilemsubj.eng
destination=c:\usr\corpora\dilsubjd.swe
meta=c:\usr\corpora\dilsubjd.met
batch=c:\usr\corpora\dilsubjd.bch

[DilemInst]
priorSubject=/Kodning/
specificSubjects=
name=Inställningsdialogen
description=CDilInstDlg är klassen för Dilemmas inställningsdialog. CDilInstDlg ärver från MFC-klassen CDialog.
language=Engelska till Svenska
source=c:\usr\corpora\dillstd.eng
destination=c:\usr\corpora\dillstd.swe
meta=c:\usr\corpora\dillstd.met
batch=c:\usr\corpora\dillstd.bch

[Flygplats]
priorSubject=/Diverse/
specificSubjects=
name=Flygplat

description=Fraser till en automat på en flygplats. Mycket Boston bli're.
language=Engelska till Svenska
source=c:\usr\corpora\boston\eng
destination=c:\usr\corpora\boston\swe
meta=c:\usr\corpora\boston\metafile
batch=c:\usr\corpora\boston\batch.txt

[RobertN]
priorSubject=/Diverse/
specificSubjects=
name=Robert Nilsson

description=Fraser ur en Robert Nilsson.
language=Engelska till Svenska
source=c:\usr\corpora\roberto.eng
destination=c:\usr\corpora\roberto.swe
meta=c:\usr\corpora\roberto.met
batch=c:\usr\corpora\roberto.bch

[/EG/]
priorSubject=/Huvudmeny/
specificSubjects=FlyttningsinomEG JordbruksinomEG MigreringfrånGrönland EtableringinomEG PrivatföretaginomEG
name=/EG/
description=Artiklar om EG

[FlyttningsinomEG]
priorSubject=/EG/
specificSubjects=
name=Flyttnings inom EG

description=Social trygghet för anställda och egenföretagare vid flyttning inom gemenskapen.
language=Engelska till Svenska
source=c:\usr\corpora\3811.eng
destination=c:\usr\corpora\3811.swe
meta=c:\usr\corpora\3811.met
batch=c:\usr\corpora\3811.bch

[EtableringinomEG]
priorSubject=/EG/
specificSubjects=
name=Etablering inom EG

description=Etableringstriben inom EG. (ursprungligen 363L0262.4)
language=Engelska till Svenska
source=c:\usr\corpora\establis.eng
destination=c:\usr\corpora\establis.swe
meta=c:\usr\corpora\establis.met
batch=c:\usr\corpora\establis.bch

[PrivatföretaginomEG]
priorSubject=/EG/
specificSubjects=
name=Privata tjänster i EG

description=Tillhandahållning av egenföretagar e inom process- och tillverknings- industrin
Jordbruk inom EG

PriorSubject=/EG/
specificSubjects=
name=Jordbruk inom EG
description=EG's allmänna handlingsprogram för upphävande av begränsningar av etableringsfriheten inom jordbruket.
language=Engelska till Svenska
source=c:/usr/corpora/process.eng
destination=c:/usr/corpora/process.swe
meta=c:/usr/corpora/process.met
batch=c:/usr/corpora/process.bch

Migrering från Grönland

PriorSubject;/EG/
specificSubjects=
name=Migrering från Grönland
description=Social trygghet för anställda vid migrering från Grönland.
language=Svenska till Engelska
source=c:/usr/corpora/process.a.swe
destination=c:/usr/corpora/process.a.swe
meta=c:/usr/corpora/process.a.met
batch=c:/usr/corpora/process.a.bch
Appendix C

BNF of the Meta file

A possible notation for context-free phrase structure grammars (CF-PSG) is the BNF (Backus-Naur Form). We have used a BNF-notation to define the syntax of Para's output and it is as follows:

\[
\begin{align*}
O &::= E,O \mid \epsilon \\
E &::= D,B,B \\
B &::= T,T \\
D &::= \text{char} \\
T &::= \text{num}
\end{align*}
\]

The non-terminals should be interpreted as:

- **E** - Represents an element in M (see section 4.2).
- **D** - Represent the hierarchical level of the element.
- **B** - Represent the boundary of a target or a source text fragment.

The first B in E represents the boundary of the text fragment in the source text. The second B represents the boundary of the corresponding text fragment in the target text. As an example, consider the following two corpora (the bold characters have the same meaning as in example 4.4.1):

**Example C1**

\[
T = "\text{Thou old, thou free, thou mountain high North; thou silent, thou beauty full of joy.} \\
\text{I greet thee, fairest land upon Earth, thy sun, thy skies, thy fields of green.}\"
\]

\[
S = "\text{Du gamla, du fria, du fjällhöga Nord,} \\
\text{du tysta, du glädjerika sköna.} \\
\text{Jag hälsar dig, vänaste land uppå Jord,} \\
\text{din sol, din himmel, dina ängder gröna.}\"
\]
The alignment will then be as follows (with two hierarchical levels):

\[ M = \text{"d,0,148,0,164,} \\
\text{p,0,10,0,10,p,10,9,10,11,p,19,19,21,26,} \\
\text{p,38,10,47,13,p,48,21,60,25,} \\
\text{p,69,16,85,14,p,85,24,99,25,} \\
\text{p,109,9,124,9,p,118,12,133,11,p,130,18,144,20"} \]

where "d" and "p" stands for document and phrase respectively.

The inherent structure of Paras' output is not shown by the BNF stated above. The elements in Paras' output is organized as a tree, where the root is the element at the highest hierarchical level:

*Figure C1 Graphical illustration of the meta tree.*
References


