Faculty of Informatics
Masaryk University

CED – Program for Corpora Editing

by

Marek Veber

FI MU Report Series **FIMU-RS-99-04**

September 1999

CED - Program for Corpora Editing

Marek Veber, `mara@fi.muni.cz`

September 10, 1999

Abstract

The article is concerned with editing of corpora, tagged corpora in particular. It introduces a corpus editor (program `ced`) and a library for work with corpora `libkorplib.a`. The following functions are described: Journaling of changes, document editing, working with a list of localities in the course of making aggregate corrections, co-operating with a corpus manager, independence of the physical data storage.

1 Introduction

The current trend towards computer processing of an ever-increasing volume of documents allows the storage of enormous text files which represent the written form of a natural language. Such files are called *text corpora.* Similarly, *phonetic corpora* can be produced from audio records. These, however, will not lie in the centre of our attention although some of the methods described below can be applied to them as well. This article will focus exclusively on text corpora. Any mention of corpora hereinafter will always refer to text corpora only.

Today, computers make it possible to process incomparably greater amounts of information than hundreds of human hands were able to process in the past. This brings about new methods for natural language processing. In the past, corpora consisted of excerpts written on paper cards which were sorted by lexicographers who then manually produced various statistics. The use of computers can considerably speed up such operations.

In order to be able to make use of the data included in the corpora, we need to be able to organize them in an efficient and logical manner, and possibly also to assign various interpretations to individual corpus segments. For this purpose, we use *tags.* These we incorporate in a corpus, either manually or automatically, by a process called tagging. Thus, *tagged (annotated) corpora* are produced.

2 Quality of the Corpora

Corpora can be used as means of deeper study of individual natural languages. However, if we want to carry out a well-informed analysis, we need as distortionless data as possible, that is to say “quality corpora”. *The quality of a corpus* depends partly on its size (but not in a linear manner) and also on the requirements on its use. For example, a corpus which contains mistakes (be it grammatical mistakes or misprints) is rather unsuitable for analysis (of grammar or morphology), but on the other hand may come in useful when constructing automatic correctors.

If we want to make the most effective use of the corpora we also need a quality instrument to manage them. By corpus management we mean the ability to add new data into a corpus, possibly also to search and alter data within it. The program systems which take care of the above-mentioned operations have been known as *corpora managers*.

The obtained data in their digital form and not yet included in a corpus will be hereinafter referred to as the *original text*. This term will not be defined any stricter for the original text is subjected to a number of inevitable [1] conversions which often bring about a considerable information omission: this concerns particularly information about the typographical features of the original document. Even data after such conversions will be referred to as the original (obtained) text. No added tags will be considered the original text. The original text which has been modified (not counting the initial conversions) will be termed more generally as the corpus text.

2.1 Corpora Managers

All the available corpus managers (such as *CQP* (see: [3])) focus above all on data search. Therefore, the corpus texts are often converted and stored in the managers in structures with numerous and complex mutual bonds. These structures may allow quick processing of inquiries, but altering the corpus text often necessitates excessively intricate modifications of the structures. This often results in the necessity to completely regenerate the data structures for the entire corpus in order to update the data after a corpus text alteration. This is why, following a corpus text alteration, the corpus managers need to have the data structures for the entire corpus all regenerated in order to update the data.

2.2 Drawbacks of the Available Instruments

The power to modify data is thus reserved for external (standard) text editors. This may well enable individual users to use their favourite editor, but the need to edit the corpus usually arises at the moment when with the help of the corpus manager the user identifies a spot which s/he wishes to modify, however, the standard editor does not have a direct link to scroll

[1]the original text must be converted into a format specific for the given corpus manager

to the identified spot. It is thus necessary to mark the given position down somehow and use the editor to locate it in the corpus text.

Users are faced with difficulties also when intending to modify the tagged corpora. The tags form a part of the corpus, however, their presence in the corpus text makes editing quite intricate. Another feature which is needed in the course of corpus editing is the ability to identify originators of individual changes and possibly to restore the original state of a corpus in the case of incorrect modifications. This facilitates correction of entered mistakes.

2.3 Motivation

We searched available resources (both literature and the Internet) for instruments which would handle corpus editing better than an "ordinary" standard editor. We arrived at the conclusion that there exist many different taggers?? and converters but the only product which can be considered a suitable instrument for corpus editing is Xcorpus[2]. However, its development has been discontinued. Surprisingly enough, we were unable to find any universal corpus editor at all.

One of the research goals of the "Laboratory for Natural Language Processing" at the Faculty of Information Science of Masaryk Univerzity in Brno is to develop the best tagged corpus possible. Corpus `DESAM` (see: [1, 2])[3] is an example of a corpus which needs revision of mistakes.

The need for a quality method of corpus tagging and of correction of mistakes led us to develop a system for corpus editing which we call *CED* (see: section. 3). The decision to create a completely new system resulted, besides other reasons, from the following:

1. the need to make corpus editing more efficient
2. non-existence of similar instruments
3. the challenge to create and modify a system according to our own specific requirements

We expect the CED system to considerably speed up the development of a tagged corpus with the number of mistakes reduced to minimum.

2.4 General Solution

Our objective is to propose a solution which will use standard resources to deal with some of the drawbacks of corpus editing (see: section. 2.2).

2.4.1 Basic Terms

Before listing the features of the proposed system, we will first describe the basic structures we will be working with. The proposed system is

[2](see: `http://www.loria.fr/projets/XCorpus/XCorpus.EN.html`)

[3]The corpus `DESAM` is composed of newspaper articles and contains about a million lexical forms. To each of the lexical forms, its morphological category and basic form are assigned, in dependence on the context.

based on structures used by the system `CQP` (see: [3]). In `CQP`, a *corpus* is composed of a list of positions among which structural tags may occur. Each *position* consists of a simple string of signs from a national alphabet which represents a basic language unit. Intuitively, an individual position can correspond to a word from a given language or to a punctuation mark, such as a question mark, a period, a quotation mark or a semicolon. Apart from the above-mentioned string, several other attributes may be assigned to each position such as: the basic form and morphological mark. By chaining individual positions we obtain a coherent corpus text. By the help of the above-mentioned structural tags, the corpus then may be divided into logical segments such as documents, paragraphs, sentences and collocations.

Our *corpus* consists of a list of *documents* where a document consists of a list of *positions*. However, the position in our system is more general than the position as defined by CQP. It does not necessarily contain a part of the corpus text. It may either correspond to the position as known from the above-mentioned description (described earlier in the text for `CQP`) or it may consist of a structural tag (unlike within `CPQ` where a structural tag is not regarded as a position). Structural tags mark a continuous block of successive positions (structure the text in a logical manner, enclose a block of positions in brackets) and assign interpretation to it in dependence on the type and attributes of a given tag.

To each position, attributes may be assigned (similarly to `CQP`) which can be interpreted also as structural tags referring only to an interval which contains just one specific position. Such attributes usually indicate the basic form (lemma), morphological category (word class, . . .) and possibly also other data specific for the given position. Similarly, each corpus and each document have their own attributes which convey further information.

Individual positions within the corpora will be unambiguously identified by the following triplet: `(kor, doc, pos)`, where `kor` defines a corpus, `doc` defines a document within the corpus and `pos` defines a position within the document. These triplets will be referred to as *localities.*

2.4.2 Journaling of changes

If we want to modify a corpus, yet keep the option to restore the original version of the text (as it was before certain changes were made or as of a certain point in time), we need the so called “versioning”. This can be handled by the process called *journaling of changes*: we keep the file containing the original text and create a file of changes where we enter individual changes made. To obtain the actual picture of the corpus, we load the original corpus into memory and gradually carry out all the changes from the file of changes. With each entered change, we note down the time and the originator of the change, regardless of whether it is a user or a program. The state before modification can be restored by skipping a given change. In the case when the skipped change deletes or inserts (a) position(s) it is necessary to re-count accordingly the changes which follow and are related to absolute numbers of positions. The described algorithm cannot, however, be implemented in practice in such a simple manner due to the enormous volume of data included in the corpora. To accelerate the process it is often inevitable to divide a corpus into shorter segments and store individual modifications of each of the segments separately in some kind of an indexed database. Changes are then loaded selectively, only for the desired corpus segment. Concrete types of changes will be dealt with later in the text (see: section. 3.2.1).

2.4.3 Effective Editing

Clear editing of the tags is made possible by means of their visual separation from the original text. Thus we separately display the corpus text, through which we scroll using the cursor, and the tags corresponding to the positioning of the cursor. This position will be hereinafter referred to as the *actual position*.

The user is allowed to scroll freely through the corpus text and at the same time watch the values of the attributes associated with a given position, while being also allowed to delete and modify individual positions, link them together or divide them and add new ones on demand. When altering attributes of a certain position, it is advisable to make use of certain external programs which identify available values from which a correct one can be selected (disambiguated).

2.4.4 Aggregate changes

Among the drawbacks suggested in chapter 2.2 belongs the failure to satisfactorily integrate standard editors into the current corpus managers.

We are often able to locate the list of localities which we intend to correct, either using the given corpus manager (for example `CQP`) or even in some other manner. What constitutes a problem is the method of scrolling through the list in a standard editor.

Therefore we first generate the list according to the given inquiry by means of an external call of the corpus manager (see: [3]). We program a function which opens the document and sets the actual position according to the given locality. This function will then facilitate scrolling through the list and editing positions determined by individual localities, and possibly their environment.

We facilitate adding a locality into the list of localities during the very editing, thus making it possible to manually tag interesting (questionable) spots in the corpus. This enables us to get back to them later.

2.4.5 Operations with Documents and Corpora

At times, a need may emerge to remove from a corpus a whole document, for example in the case of its unfitness or duplicity. In such a case, editing of the whole corpus may be feasible neither timewise, nor memory-wise. Therefore we make use of a value of the document's attribute (hereinafter referred to as `DEL`). The value will be set in such a manner so as to distinguish between the documents intended for deletion and those which are to be kept. For example, for the documents to be removed we set `DEL = 1`, otherwise `DEL = 0`. In addition, we assign a special attribute `EXPR` to the corpus whose value will be represented by a logical expression (condition). The condition will be met in the case of the documents which are to be retained in the corpus. The rest of the documents will be ignored in the subsequent operations. That is for example `EXPR="DEL==0"`.

3 The CED System

We have developed a special program instrument for corpus editing which we call **CED** - an abbreviation for **C**orpora **Ed**itor. It is compatible with any operating system based on UNIX, DOS or MS-WINDOWS. The whole system is implemented in the "C++" language. It consists of a library for work with the corpus `libkorplib.a`, a display library `libcase.a` and a program written using these two libraries (`ced`, or `ced.exe` for DOS and MS-WINDOWS).

The system implements all the features described in chapter 2.4.

3.1 The Library for Work with the Corpora

The creation of this library was essential for the development of the program for corpus editing `ced`. It allows a certain level of abstraction from physical implementation, which will be henceforth referred to as the application level. Thus, parallel access to corpora with mutually different physical data storage is made possible, one needs only to have access to implementations (see: section. 3.2) for the given storage types.

3.2 Physical Implementation

To date, we have only implemented and described the method of accessing the corpus data stored in text files (see below). In the future, we expect to introduce two more extensions: to the data stored in the SQL database (for example `postgres` (see: [8]) or `mSQL` (see: [9]))) and to the data provided by the client-server system via network. This system links one of the above-mentioned variants of physical implementation with the application level, when the two levels are separated by the network.

The advantage of the SQL database over text files will lie in the ability to process faster large corpora and use the network to distribute data and interconnect various platforms.

Text Files: A combination of text files and a library is applied here. *** The library uses the *** mechanism to associate a certain item (key) with another one that contains the entered data. The file managed by the library will be referred to as the gdbm-index.

A file containing a *list of documents* is used for the list of corpora and their attributes. It is a gdbm-index which serves to associate a corpus number with its attributes. The corpus attributes provide the following information: corpus name, path to the main corpus file and its index, and possibly also paths to the files of changes (see below).

The main text file includes the original corpus text. Individual positions (see: section. 2.4.1) are separated by new-line markers and attributes in each position are separated by tabulator markers. Boundaries between individual documents are marked by structural tags.

To each main file a gdbm-index is assigned which serves to associate a document number with the corresponding documents' attributes. One of these attributes indicates the document's initial position (offset) in the main file. The whole document can thus be accessed using the document number.

A *file of changes* is a text file which contains one change on each line. To the file of changes a gdbm-index is assigned which associates a document number with the position (in the text file) of the first and the last change relevant for this document. Each change consists of several items separated either by the character ":" or "$". The first item has a fixed length (10 digits) and indicates the position of the next change for the same document in the same file of changes. The value 0 marks the last item of the list. This item, together with the relevant gdbm-index, serves only to create lists of changes for individual documents. The second item indicates the number of the document to which the change applies. The third item is an identification number of the user, or the program, who carried out the change. The fourth one introduces a list of commands (see: section. 3.2.1) separated by semicolons, for this change.

An example of changes for the 1st document made by a user number 11587 which set the value of the lemma attribute of the first position in the document to "nemocnice"[4] and the value of the tag attribute of the zero position to "k2"[5]:

```
0000000095$1:915707012:11578:[1].L="nemocnice";
0000000000$1:915707013:11578:[0].T="k2";
```

3.2.1 Changes in the Documents

We will enter the notation for the entry of corrections which are to be carried out (commands for corrections). In the course of editing a document composed of positions ***, the following elementary operations will be sufficient: entering a new position into a specific place in the list of positions, deleting a specific position and altering a position's attribute.

For some purposes, it is advisable to create abbreviations for certain sequences consisting of the above mentioned elementary operations. The

[4]hospital
[5]adjective

following will be used: linking several positions into one, dividing one position into two, demarking a segment of positions by a structural tag.

We use the following notation:

- `I(pos)` — insert a new position after the actual one (-1 means before the very first position), all attributes of the new position are empty
- `D(from, len)` — clear the given interval of positions
- `[pos].A="value"` — set attribute `A` of position `pos` to given value, where `A` is either serial number of the attribute or attribute's symbolic name: `0` $\equiv$ `W` (word), `1` $\equiv$ `L` (lemma), `2` $\equiv$ `T` (tag), `3` $\equiv$ `N` (note)
- `J(from, len)` — join the interval of positions to one position, store new value of attribute `W` as a chain of `W` attributes for all joined positions, other attributes of the newly created position are empty.
- `S(pos, col)` — split position in two, both have only the value of `W` attribute set (other attributes are empty), new values of `W` attributes are created by splitting of `W` attribute of the original position after `col`-th character
- `Q(from, len, "zn arg1 arg2 ...")` — mark the interval of positions with given structural tag

All changes in documents will be provided by the mentioned commands. A journal of changes will be produced, this journal enables versioning of corpora's data.

4 Conclusions

We have developed system for easy corpora editation (program ced). This system can be used on UNIX terminals or with X-WINDOWS, on DOS and MS-WIDOWS platforms. A special graphics library is used by the system, this library provides interactive tools such as: menu, hot-keys, dialog windows, etc.

System enables to work with list of localities, that can be created in several ways. A list of localities can be explored (step by step through a part of the given list) and the corpora positions denoted by localities in a corpus can be repaired, if needed. This approach enables making of aggregate changes in localities selected using complicated queries[6].

All changes made in corpora are logged in journal to enable determining the author of the change and enable undoing selected changes.

This program has been used to mark sentence boundaries in the corpus `DESAM` (see: [1, 2]) and other aggregate changes.

[6]by external programs `cqp` (see: [3])

References

[1] K. Pala, P. Rychlý, and P. Smrž. DESAM — Approaches to Disambiguation. Technical Report FIMU-RS-97-09, Faculty of Informatics, Masaryk University, Brno, 1997.

[2] K. Pala, P. Rychlý, and P. Smrž. DESAM — Annotated Corpus for Czech. In *Proceedings of SOFSEM'97*. Springer-Verlag, 1997.

[3] *CQP* B.Maximilian, O.Christ — The CQP User's Manual, Universität Stuttgart, 1994.

[4] Philip A. Nelson — The GNU database manager, Free Software Foundation, Inc., (see: `http://www.gnu.org`)

[5] P.Ševeček : *LEMMA* — morphological analyzer and lemmatizer for Czech, program in "C", Brno, 1996. (manuscript).

[6] V. Puža — Diploma Thesis, Faculty of Informatics, Masaryk University, Brno, duben 1997.

[8] `http://www.cz.postgresql.org`.

[9] `http://www.Hughes.com.au/products/msql`.