# Towards a Dedicated Database Management System for Dictionaries

Marc Domenig, Patrick Shann
Institut Dalle Molle pour les Etudes
Semantiques et Cognitives (ISSCO)
Route des Acacias 54
1227 GENEVA, Switzerland

**Abstract**

This paper argues that a lexical database should be implemented with a special kind of database management system (DBMS) and outlines the design of such a system. The major difference between this proposal and a general purpose DBMS is that its data definition language (DDL) allows the specification of the entire morphology, which turns the lexical database from a mere collection of 'static' data into a real-time word-analyser. Moreover, the dedication of the system conduces to the feasibility of user interfaces with very comfortable monitor- and manipulation functions.

## 1. Introduction

As the means of natural language processing are gradually reaching a stage where the realisation of large-scale projects like EUROTRA becomes more and more feasible, the demand for lexical databases increases. Unfortunately, this is not a demand which is easy to meet, because lexical databases are exceedingly expensive. The two main reasons for this are the following:

- The manual labour involved with the coding of entries is time-consuming.
- The possibilities to take over or to cumulate existing machine-readable dictionaries are rather limited because existing dictionaries usually contain only a part of the information needed for a certain project. Severe consistency problems and the need for manual post-editing are the result of this (->[Hess, et. al. 1983]).

As long as there is no general agreement on the kind of information which should be stored in a dictionary and therefore no universally applicable lexical database, we will have to live with these problems. The important question for the time being is, whether we can alleviate them. This paper argues that the best way to do that is to construct a *dedicated* database management system (DBMS). It presents a prototype proposal which has been conceived in a doctoral thesis [Domenig 1986] and which is the basis for a project that ISSCO[1] has recently started in conjunction with the Swiss National Fund. Because of the limited space at disposal we will mainly explain the most uncommon feature of the system, its morphological capabilities. We will not go into all of the monitor- and manipulation functions which alleviate the task of lexicography. The reader may infer the potential for them, however, if he remembers the following fact: as both the 'static' and 'dynamic' informations about entries (features and morphological processes, respectively) are coded *within* the system, they can both be *accessed and controlled* quite easily.

## 2. The requirements for a lexical database

According to our opinion, a lexical database should not be a mere collection of 'static' data, i.e. a set of morphemes with associated features. It should comprise *morphological processes* which enable it to serve as a real-time word-analyser used in a message-switching environment (e.g. a local area network). Moreover, the DBMS should control the consistency of the data as far as possible so that only *plausible* combinations of features and morphological processes can be associated with entries. This differs very much from the 'traditional' concept of lexical databases, where the entries consist of strings with associated features and the morphological interpretation is done *outside* of the database in a program. Naturally, the control over consistency is much more efficient and also easier to maintain if both 'static' and 'dynamic' information are coded within the database.

1. ISSCO stands for 'Institut dalle Molle pour des Etudes Semantiques et Cognitives'.

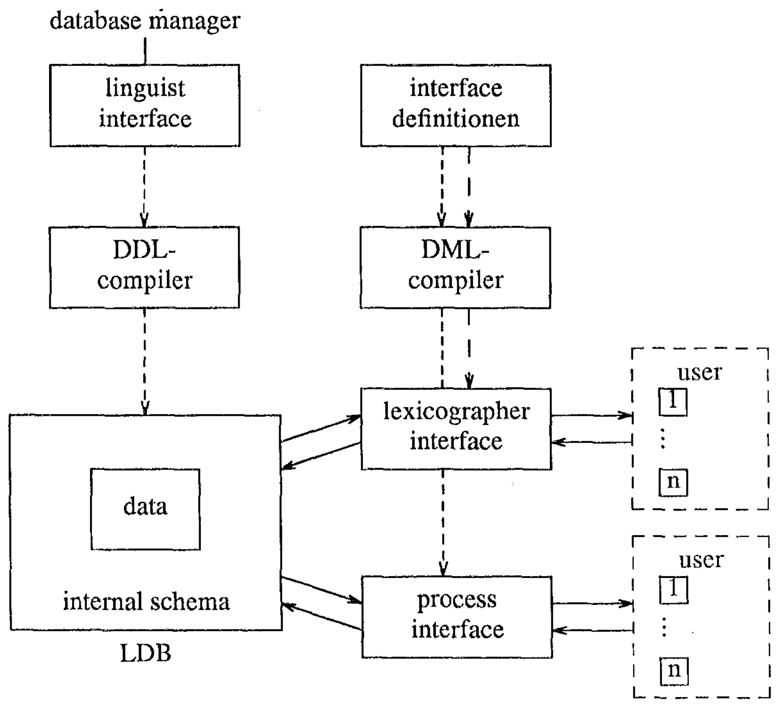


Fig. 1: The DBS according to the ANSI/SPARC 3-Schema-Model

## 3. The inadequacy of general purpose DBMS

General purpose DBMS - be they relational or whatever - do not live up to the formulated requirements for a real-time dictionary database. On the one hand, they are in many areas much *too powerful* for the task at hand, i.e. they can be adapted to a wealth of problems which have nothing to do with dictionaries. This flexibility ensues both a relatively low level of abstraction and a massive overhead. On the other hand, general purpose DBMS are *not powerful enough;* for example, a relational data definition language (DDL) provides no transparent means to express morphological processes.

## 4. The design of the dedicated DBMS

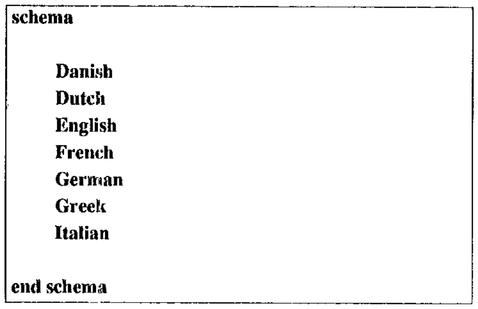
The design of the dedicated DBMS put forward in [Domenig 1986] follows the ANSI/SPARC 3-Schema-Model. As shown in Fig. 1, it assumes that three different interfaces are needed:

- A *linguist interface* with which the *conceptual schema* is defined, i.e. the structure and the consistency rules of the database.
- A *lexicographer interface* for the editing of entries.
- A *process interface* for the real-time question-answering service in the message-switching environment.

From the point of view of the software-design, the most complex part of this conception is the linguist interface with the DDL and its compiler. All the other parts of the system depend very much on it because of its far-reaching dedication. We will therefore concentrate on the linguist interface and the DDL in this paper. The principal guidelines for their definition have been the following:

- The syntax of the DDL should be intelligible for linguists.
- The linguist interface should be interactive and give some leeway for experiments in order to test different morphological strategies.

The proposed solution foresees the implementation of the system on a high-performance workstation. It includes multiple window technology with pop-up menus for monitor- and manipulation functions as well as incremental compilation. Some brief examples: The top-level window of the interface looks as follows (if we assume that we have seven dictionaries):

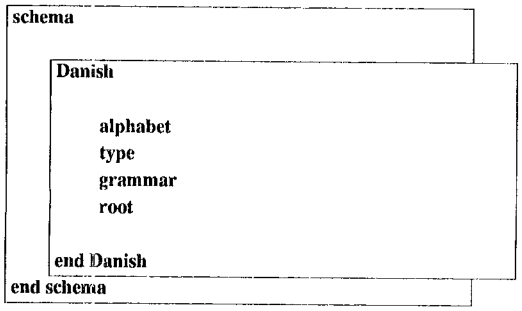


```
schema

    Danish
    Dutch
    English
    French
    German
    Greek
    Italian

end schema
```

If the linguist wants to define the conceptual schema of the Danish dictionary he selects - with a mouse - the according string on the screen, whereupon a second window is pasted on top of the existing one:

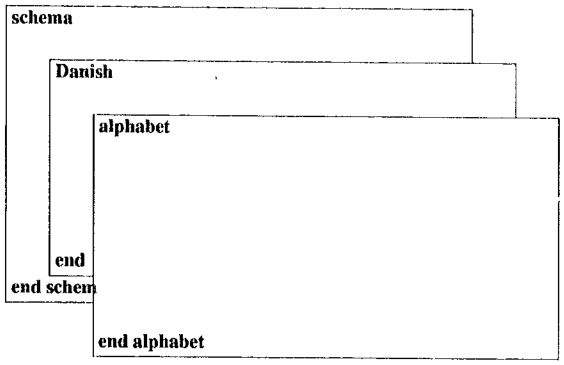


```
schema
    Danish

        alphabet
        type
        grammar
        root

    end Danish
end schema
```

Identically to the top-level window, this window is unalterable, i.e. all the dictionary schemas consist of four different definition parts, an *alphabet-*, a *type-*, a *grammar-* and a *structure*-definition (the structure-definition is represented by the keyword *root*). If the linguist wants to edit one of the definition parts, he again selects the according string:



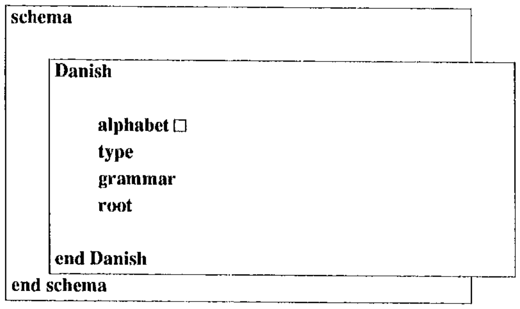
```
schema
    Danish
        alphabet



    end
end schem


        end alphabet
```

In contrast to the two top-levels, this window can be edited. We will not go into the function or the syntax of the alphabet-definition as both are quite trivial. As might be inferred from the name, this is the place where character sets and the like are defined (because the system discerns a lexical and a surface level, some metacharacters denoting morphological classes etc., the character set is not quite as trivial as might be imagined at first glance). If something is entered into this window, the according string in the window *above* appears henceforth with an icon (□) behind it:



```
schema
    Danish

        alphabet □
        type
        grammar
        root

    end Danish
end schema
```

In a similar fashion the other three definition parts of the conceptual schema can be defined: The *type* definition comprises name- and domain-specifications of all but the string-typed features allowed in the database. We will not go into its syntax here either.

The grammar definition contains morphonological rules which mediate between the lexical and the surface level. We have adapted their concept from *Koskenniemi* ([Koskenniemi 1983, 1984]), whose formalism has been widely acknowledged by now, especially in the US (at SRI [Shieber 1984], University of Texas [Karttunen 1983], by M. Kay of Xerox etc.). A few examples:

**example 1: epenthesis**

$$\mathit{rule}:\ +/e \longleftrightarrow \left[ \left[ c \mid s\,(h) \right] \mid S \mid y/i \right] \_\ s$$

**example 2: consonant-doubling**

$$\mathit{rule}:\ +/{<}C1{>} \longleftrightarrow \left[ ` \mid \# \right] C^{*}\ V {<}C1{>} \_\ V$$

$$\mathit{where}\ {<}C1{>} = \{b, d, f, g, l, m, n, p, r, s, t\}$$

**example 3: surface-'i' for lexical 'y'**

$$\mathit{rule}:\ y/i \longleftrightarrow C \_\ +/= \wedge \left[ i \mid a \right]$$

**example 4: elision**

```
rule: e/0 <—>      <C2> _ +/0 V,
                    ^V V _ +/0 e,
                   [g | c] _ +/0 [e | i]

where <C2> = {CP; CP in ^V & CP in ^{c, g} }
```

**example 5: surface-'y' for lexical 'i'**

```
rule: i/y <—> _ e/0 +/0 i
```

The structure definition is at least syntactically the most complex part of the conceptual schema. It contains an arbitrary number of hierarchical levels which define a collection of so called *lexical unit classes* (luclasses) on the one hand, *irregular entries* (luentries) on the other. The fundamental ideas behind it are:

- Entries which obey the same morphological rules should be grouped into *classes* so that those rules have to be specified only once.
- Entries which are *too irregular* to fit into such a class should be defined as *irregular*. The *boundary* between regularity/irregularity should be defined by the database manager (linguist) and hence be unalterable by lexicographers. Irregular entries are therefore defined in the conceptual schema (the interactivity of the interface, the powerful editing functions and the incremental compilation provide for the feasibility of this approach).

The consequence of this approach is that the structure definition consists of a set of *luclass*-definitions on the one hand, a set of *luentry*-definitions on the other. In order to facilitate the management of the members of these sets, they are organized in a hierarchical structure, whereas the criteria for the hierarchy are *some of the features which qualify* the sets. Syntactically, this looks e.g. as follows:

```
root
    dcl □
    gcase □
    dcase
        [{Cat:N,      node □} |
         {Cat:V,      node □} |
         {Cat:ADJ,    node □} |
         {Cat:ADV,    node □} |
         {Cat:AUX,    node □} |
         {Cat:DET,    node □} |
         {Cat:PRO,    node □} |
         {Cat:PRE,    node □} |
         {Cat:CON,    node □} |
         {Cat:INT,    node □}]
    end dcase
end root
```

This window defines one hierarchical level (the top) of the organization of the luclasses and luentries respectively. The meaning of it should be quite obvious if we leave out **dcl □** and **gcase □** and concentrate on the case-distinction enclosed in the square brackets: The features **Cat:N, Cat:V,...** are defined to be distinctive for certain subsets out of the collection of luclasses and luentries. Note that the names of the attributes and values are entirely arbitrary (they must be defined in the type-definition, of course). Subordinate levels of the definition are again abstracted by icons (**node □**), i.e. they are defined and viewed in separate windows:

```
root
    dcl □
    gcase □
    dcase
        [{Cat:N,      node □} |
         {Cat:V,     node
         {Cat:ADJ,       dcl □
         {Cat:ADV,       gcase □
         {Cat:AUX,       dcase
         {Cat:DET,           [{VCat:REG, node □} |
         {Cat:PRO,            {VCat:IRREG, node □}]
         {Cat:PRE,       end dcase
         {Cat:CON,   end node
         {Cat:INT, node □}]
    end dcase
end root
```

In the leaves of this tree the keyword **node** is replaced by either **luclass** or **luentry**. Their syntax is almost identical, so let us just give an example of an **luclass**-definition:

```
luclass
      trans□
      gcase□

      Fenster              + [  {Case:NOM, Number:SG} |
                                {Case:NOM, Number:PL} |
                                {Case:DAT, Number:SG}  |
                                {Case:GEN, Number:PL}  |
                                {Case:AKK, Number:SG}  |
                                {Case:AKK, Number:PL}   ]

                   +s      +    {Case:GEN, NUMBER:SG}

                   +n      +    {Case:DAT, NUMBER:PL}
end luclass
```

Apart from the strings **trans□** and **gcase□**, the meaning of it should again be quite obvious. In prose we might summarize it as follows: All entries of this class are nouns of a certain subclass - the features **Cat:N,...** denoting this qualification are specified on the path from the root to this leaf - and within this subclass a zero-morpheme attached to the stem is interpreted as one of the following alternatives of feature sets:

```
[ {Case:NOM, Number:SG} |
  {Case:NOM, Number:PL} |
  {Case:DAT, Number:SG}  |
  {Case:GEN, Number:PL}  |
  {Case:AKK, Number:SG}  |
  {Case:AKK, Number:PL}   ]
```

An 's'-morpheme attached to the stem is interpreted as **{Case:GEN, NUMBER:SG}**, an 'n'-morpheme as **{Case:AKK, Number:PL}**. The string **Fenster** acts in this definition mainly as an illustrative *example*, i.e. it has no conceptual function and may be replaced by all noun-stems belonging to this class. Conceptually speaking, the definition therefore specifies all the inflectional forms of this noun class. The consequence of this is that lexicographers have to enter only the stems of words, the inflections are defined in the system. Together with some additional language constructs, the regularities of morphology can thus be quite thoroughly grasped. The additional constructs are:

- a formalism with approximately the power of a context-free grammar for compounding and derivation which allows the combination of different luclasses and luentries.
- a formalism for the specification of stem-alterations (e.g. German Umlaut).

## 5. Conclusion

The important difference of this approach compared to other systems is the definition of morphological phenomena in the conceptual schema of the DBMS itself. This conceptual schema can be easily compiled into a redundancy-optimized *internal schema*. This in turn provides for two things: first for an efficient real-time access to the lexical units etc., second for very comfortable monitor- and manipulation-functions for the linguist interface. For example, it is trivial to implement functions which generate all forms which are associated with certain features or combinations thereof. It is equally easy to test the impact of complex rules, be they grammar-rules of the Koskenniemi-style or difficult to handle compounding rules (implemented by the formalism which is similar to a context-free grammar). The most intriguing quality of the internal schema, however, is probably that it enables the database manager (linguist) to alter the morphological strategies dynamically, i.e. to experiment with them. This is possible, because the system always knows which syntactico-semantic features and which morphological rules have to be associated with the different classes of entries; whenever those associations - you could also call them consistency rules - are altered, the system can determine whether the entries belonging to the according classes lose or gain information, whether the alteration is legal etc.. We do not want to go further into those consistency problems as we have not really explained them in this summary. We would like to stress, however, that we consider their integration in the DBMS a major advantage and necessity as they *autonomize* the whole system. Apart from the possibilities for experiments they facilitate the integration of existing machine-readable dictionaries, again, because the system always knows which kind of information is distinctive and which is mandatory for which class of entries.

Summarising we could say that the kind of morphology supported by the DBMS is rather a traditional one, i.e. the biggest effort has been spent on truly regular phenomena like inflection. For compounding and derivation the offered choice is either a full implementation (-->redundancy) or the rather dangerous - potentially overgenerating - formalism resembling a context-free grammar. It has to be stressed that we conceive this system as a *prototype* which will probably be subject to some alterations in the future. The proposed software-design is accordingly tuned, i.e. it relies on the availability of powerful software tools (EMACS, LEX, YACC, LISP etc.) running in a UNIX-environment.

## 6. References

Amsler R.A.: "Machine-Readable Dictionaries." *Annual Review of Information Science and Technology (ARIST)*, Vol. 19, 1984, 161-209.

ANSI/X3/SPARC Study Group on Data Base Management Systems: "Interim Report 75-02-08." *FDT (Bull. of the ACM SIGMOD)* 7, 1975.

Cignoni L., et al. eds: *Survey of Lexicographic Projects*. Istituto di Linguistica Computazionale, Pisa, Italy, 1983.

Domenig M.: *Entwurf eines dedizierten Datenbank-systems für Lexika.* Doctoral thesis at the University of Zürich, in print.

Hess K., Brustkern J., Lenders W.: *Maschinenlesbare deutsche Wörterbücher.* Niemeyer, Tübingen, 1983.

Jaspaert L.: *Matters of Morphology. EUROTRA Morphology Legislation, Second Draft.* Internal EUROTRA paper, Nov. 1984.

Karttunen L.: "KIMMO: A Two Level Morphological Analyzer." *Texas Linguistic Forum* 22, 1983, 165-186.

Koskenniemi K.: *Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production.* Doctoral thesis at the University of Helsinki, Department of General Linguistics, Publications No. 11, 1983.

Koskenniemi K.: "A General Model for Word-Form Recognition and Production." in *Proceedings: 10th International Conference on Computational Linguistics,* Stanford Univ., Calif., July 2-6, 1984.