Emdros – a text database engine for analyzed or annotated text

Ulrik Petersen
Department of Communication
University of Aalborg
Kroghstræde 3
DK – 9220 Aalborg East
Denmark
ulrikp@hum.aau.dk

Abstract

Emdros is a text database engine for linguistic analysis or annotation of text. It is applicable especially in corpus linguistics for storing and retrieving linguistic analyses of text, at any linguistic level. Emdros implements the EMdF text database model and the MQL query language. In this paper, I present both, and give an example of how Emdros can be useful in computational linguistics.

1 Introduction

As (Abeillé, 2003) points out, "corpus-based linguistics has been largely limited to phenomena that can be accessed via searches on particular words. Inquiries about subject inversion or agentless passives are impossible to perform on commonly available corpora" (p. xiii).

Emdros is a text database engine which attempts to remedy this situation in some measure. Emdros' query language is very powerful, allowing the kind of searches which Abeillé mentions to be formulated quickly and intuitively. Of course, this presupposes a database which is tagged with the data necessary for answering the query.

Work has been done on supporting complex queries, e.g., (Bird et al., 2000; Cassidy and Bird, 2000; Mengel, 1999; Clarke et al., 1995). Emdros complements these pieces of work, providing a working implementation of many of the features which these systems support.

In this paper, I present the EMdF text database model on which Emdros rests, and the MQL query language which it implements. In addition, I give an example of how Emdros can be useful in answering questions in computational linguistics.

2 History of Emdros

Emdros springs out of a reformulation and implementation of the work done by Crist-Jan Doedens in his 1994 PhD thesis (Doedens, 1994). Doedens defined the MdF (Monads-dot-Features) text database model, and the QL query language. Doedens gave a denotational semantics for QL and loaded QL with features, thus making it very difficult to implement. The present author later took Doedens' QL, scaled it down, and gave it an operational semantics, hence making it easier to implement, resulting in the MQL query language. I also took the MdF model and extended it slightly, resulting in the EMdF model. Later, I implemented both, resulting in the Emdros text database engine, which has been available as Open Source software since October 2001. The website[1] has full sourcecode and documentation.

Emdros is a general-purpose engine, not a specific application. This means that Emdros must be incorporated into a specific software application before it can be made useful.

3 The EMdF model

The EMdF model is an extension of the MdF model developed in (Doedens, 1994). The EMdF (Extended MdF) model is based on four concepts: Monad, object, object type, and feature. I describe each of these in turn, and give a small example of an EMdF database.

3.1 Monad

A monad is simply an integer. The sequence of the integers (1,2,3, etc.) dictates the sequence of the text. The monads do not impose a *reading-direction* (e.g., left-to-right, right-to-left), but merely a *logical text-order*.

3.2 Object

An object is simply a set of monads with an associated object type. The set is arbitrary in the sense that there are no restrictions on the set. E.g., {1}, {2}, {1,2}, {1,2,6,7} are all valid objects. This allows for objects with gaps, or discontiguous objects (e.g., discontiguous clauses). In addition, an object always has a unique integer id, separate from the the object's monad set.

Objects are the building blocks of the text itself, as well as the annotations or analyses in the

[1] `http://emdros.org/`

database. To see how, we must introduce object types.

3.3 Object type

An object type groups a set of objects into such classes as "Word", "Phrase", "Clause", "Sentence", "Paragraph", "Chapter", "Book", "Quotation", "Report", etc. Generally, when designing an Emdros database, one chooses a *monad-granularity* which dictates the smallest object in the database which corresponds to one monad. This smallest object is often "Word", but could be "Morpheme", "Phoneme" or even "Grapheme". Thus, for example, Word number 1 might consist of the object set $\{1\}$, and Word number 2 might consist of the object set $\{2\}$, whereas the first Phrase in the database might consist of the set $\{1,2\}$.

3.4 Feature

An object type can have any number of *features*. A feature is an attribute of an object, and always has a type. The type can be a string, an integer, an enumeration, or an object id. The latter allows for complex interrelationships among objects, with objects pointing to each other, e.g., a dependent pointing to a head.

An enumeration is a set of labels with values. For example, one might define an enumeration "psp" (part of speech) with labels such as "noun", "verb", "adjective", etc. Emdros supports arbitrary definition of enumeration label sets.

3.5 Example

Consider Figure 1. It shows an EMdF database corresponding to one possible analysis of the sentence "The door was blue." There are three object types: Word, Phrase, and Clause. The Clause object type has no features. The Phrase object type has the feature "`phr_type`" (phrase type). The Word object type has the features "`surface`" and "`psp`".

The monad-granularity is "Word", i.e., each monad corresponds to one monad. Thus the word with id 10001 consists of the monad set $\{1\}$. The phrase with id 10005 consists of the monad set $\{1,2\}$. The single clause object consists of the monad set $\{1,2,3,4\}$.

The text is encoded by the "`surface`" feature on Word object type. One could add features such as "`lemma`", "`number`", "`gender`", or any other feature relevant to the database under construction. The Phrase object type could be given features such as "`function`", "`apposition_head`", "`relative_head`", etc. The Clause object type could be given features distinguishing such things as "VSO order", "tense of verbal form", "illocutionary force", "nominal clause/verbless clause", etc. It all depends on the theory used to describe the database, as well as the research goals.

	1	2	3	4
word	**w: 10001** surface: The psp: article	**w: 10002** surface: door psp: noun	**w: 10003** surface: was psp: verb	**w: 10004** surface: blue. psp: adjective
phrase	**p: 10005** phr_type: NP		**p: 10006** phr_type: VP	**p: 10007** phr_type: AP
clause	**c: 10008**			

Figure 1: A small EMdF database

4 The MQL query language

MQL is based on two properties of text which are universal: sequence and embedding. All texts have sequence, dictated by the constraints of time and the limitation of our human vocal tract to produce only one sequence of words at any given time. In addition, all texts have, when analyzed linguistically, some element of embedding, as embodied in the notions of phrase, clause, sentence, paragraph, etc.

MQL directly supports searching for sequence and embedding by means of the notion of *topographicity*. Originally invented in (Doedens, 1994), a (formal) language is topographic if and only if there is an isomorphism between the structure of an expression in the language and the objects which the expression denotes.

MQL's basic building block is the *object block*. An object block searches for objects in the database of a given type, e.g., Word, Phrase or Clause. If two object blocks are adjacent, then the objects which they find must also be adjacent in the database. If an object block is embedded inside another object block, then the inner object must be embedded in the outer object in the database.

Consider Figure 2. It shows two adjacent object blocks, with feature constraints. This would find two Phrase objects in the database where the first is an NP and the second is a VP. The objects must be adjacent in the database because the object blocks are adjacent.

```
[Phrase phrase_type = NP]
[Phrase phrase_type = VP]
```

Figure 2: Two adjacent object blocks

Now consider Figure 3. This query would find a clause, with the restriction that embedded inside the clause must be two phrases, a subject NP and

a predicate VP, in that order. The "`..`" operator means that space is allowed between the NP and the VP, but the space must be inside the limits of the surrounding clause. All of this presupposes an appropriately tagged database, of course.

```
[Clause
  [Phrase phrase_type = NP
          and function = Subj]
  ..
  [Phrase phrase_type = VP
          and function = Pred]
]
```

Figure 3: Examples of embedding

The restrictions of type "`phrase_type = NP`" refer to features (or attributes) of the objects in the database. The restriction expressions can be any Boolean expression (and/or/not/parentheses), allowing very complex restrictions at the object-level.

Consider Figure 4. It shows how one can look for objects inside "gaps" in other objects. In some linguistic theories, the sentence "The door, which opened towards the East, was blue" would consist of one discontiguous clause ("The door ... was blue") with an intervening nonrestrictive relative clause, not part of the surrounding clause. For a sustained argument in favor of this interpretation, see (McCawley, 1982). The query in Figure 4 searches for structures of this kind. The surrounding context is a Sentence. Inside of this sentence, one must find a Clause. The first object in this clause must be a subject NP. Directly adjacent to this subject NP must be a *gap* in the surrounding context (the Clause). Inside of this gap must be a Clause whose clause type is "`nonrestr_rel`". Directly after the close of the gap, one must find a VP whose function is predicate. Mapping this structure to the example sentence is left as an exercise for the reader.

```
[Sentence
  [Clause
    [Phrase FIRST phrase_type = NP
       and function = Subj]
    [gap
      [Clause cl_type = nonrestr_rel]
    ]
    [Phrase phrase_type = VP
            and function = Pred]
  ]
]
```

Figure 4: An example with a gap

Lastly, objects can refer to each other in the query. This is useful for specifying such things as agreement and heads/dependents. In Figure 5, the "`AS`" keyword gives a name ("`w1`") to the noun inside the NP, and this name can then be used inside the adjective in the AdjP to specify agreement.

```
[Phrase phrase_type = NP
  [Word AS w1 psp = noun]
]
[Phrase phrase_type = AdjP
  [Word psp = adjective
      and number = w1.number
      and gender = w1.gender]
]
```

Figure 5: Example with agreement

MQL provides a number of features not covered in this paper. For full documentation, see the website.

The real power of MQL lies in its ability to express complex search restrictions both at the level of structure (sequence and embedding) and at the object-level.

5 Application

One prominent example of an Emdros database in use is the Werkgroep Informatica (WI) database of the Hebrew Bible developed under Prof. Dr. Eep Talstra at the Free University of Amsterdam. The WI database is a large text database comprising a syntactic analysis of the Hebrew Bible (also called the Old Testament in Hebrew and Aramaic). This is a 420,000 word corpus with about 1.4 million syntactic objects. The database has been analyzed up to clause level all the way through, and has been analyzed up to sentence level for large portions of the material. A complete description of the database and the underlying linguistic model can be found in (Talstra and Sikkel, 2000).

In the book of Judges chapter 5 verse 1, we are told that "Deborah and Barak sang" a song. Deborah and Barak are clearly a plural entity, yet in Hebrew the verb is feminine singular. Was this an instance of bad grammar? Did only Deborah sing? Why is the verb not plural?

In Hebrew, the rule seems to be that the verb agrees in number and gender with the first item in a compound subject, when the verb precedes the subject. This has been known at least since the 19th century, as evidenced by the Gesenius-Kautzsch grammar of Hebrew, paragraph 146g.

With Emdros and the WI database, we can validate the rule above. The query in Figure 6 finds

234 instances, showing that the pattern was not uncommon, and inspection of the results show that the verb most often agrees with the first member of the compound subject. The 234 "hits" are the bare results returned from the query engine. It is up to the researcher to actually look at the data and verify or falsify their hypothesis. Also, one would have to look for counterexamples with another query.

```
[Clause
  [Phrase function = Pred
    [Word AS w1 psp = verb
          and number = singular]
  ]
  ..
  [Phrase function = Subj
    [Word (psp = noun
       or psp = proper_noun
       or psp = demonstrative_pronoun
       or psp = interrogative_pronoun
       or psp = personal_pronoun)
      and number = singular
      and gender = w1.gender]
    ..
    [Word psp = conjunction]
  ]
]
```

Figure 6: Hebrew example

The query finds clauses within which there are two phrases, the first being a predicate and the second being a subject. The phrases need not be adjacent. The predicate must contain a verb in the singular. The subject must first contain a noun, proper noun, or pronoun which agrees with the verb in number and gender. Then a conjunction must follow the noun, still inside the subject, but not necessarily adjacent to the noun.

The WI database is the primary example of an Emdros database. Other databases stored in Emdros include the morphologically encoded Hebrew Bible produced at the Westminster Hebrew Institute in Philadelphia, Pennsylvania, and a corpus of 67 million words in use at the University of Illinois at Urbana-Champaign.

6 Conclusion

In this paper, I have presented the EMdF model and the MQL query language as implemented in the Emdros text database engine. I have shown how MQL supports the formulation of complex linguistic queries on tagged corpora. I have also given an example of a specific problem in Hebrew linguistics which is nicely answered by an Emdros query. Thus Emdros provides a solid platform on which to build applications in corpus linguistics, capable of answering linguistic questions of a complexity higher than what most systems can offer today.

Acknowledgements

My thanks go to Constantijn Sikkel of the Werkgroep Informatica for coming up with the problem for the Hebrew query example.

References

Anne Abeillé. 2003. Introduction. In Anne Abeillé, editor, *Treebanks – Building and Using Parsed Corpora*, volume 20 of *Text, Speech and Language Technology*, pages xiii–xxvi. Kluwer Academic Publishers, Dordrecht, Boston, London.

Steven Bird, Peter Buneman, and Wang-Chiew Tan. 2000. Towards a query language for annotation graphs. In *Proceedings of the Second International Conference on Language Resources and Evaluation*, pages 807–814. European Language Resources Association, Paris. http://arxiv.org/abs/cs/0007023.

Steve Cassidy and Steven Bird. 2000. Querying databases of annotated speech. In *Database technologies: Proceedings of the Eleventh Australasian Database Conference*, pages 12–20. IEEE Computer Society.

Charles L. A. Clarke, G. V. Cormack, and F. J. Burkowski. 1995. An algebra for structured text search and a framework for its implementation. *The Computer Journal*, 38(1):43–56.

Christianus Franciscus Joannes Doedens. 1994. *Text Databases: One Database Model and Several Retrieval Languages*. Number 14 in Language and Computers. Editions Rodopi Amsterdam, Amsterdam and Atlanta, GA.

James D. McCawley. 1982. Parentheticals and discontinuous constituent structure. *Linguistic Inquiry*, 13(1):91–106.

Andreas Mengel. 1999. MATE deliverable D3.1 – specification of coding workbench: 3.8 improved query language (Q4M). Technical report, Institut für Maschinelle Sprachverarbeitung, Stuttgart, 18 Nov. http://www.ims.uni-stuttgart.de/projekte/mate/q4m/.

Eep Talstra and Constantijn Sikkel. 2000. Genese und Kategorienentwicklung der WIVU-Datenbank. In Christof Hardmeier, Wolf-Dieter Syring, Jochen D. Range, and Eep Talstra, editors, *Ad Fontes! Quellen erfassen - lesen - deuten. Was ist Computerphilologie?*, volume 15 of *APPLICATIO*, pages 33–68, Amsterdam. VU University Press.