Emdros: The database engine for analyzed or annotated text

created by Ulrik Sandborg-Petersen

Reviewed by Kirk E. Lowery, Westminster Theological Seminary, Philadelphia

1. INTRODUCTION. Emdros (http://emdros.org) is an open source search engine with a powerful query language (MQL) optimized for linguistically analyzed or arbitrarily annotated text, offering a choice of backends (presently PostgreSQL, MySQL, SQLite 2, or SQLite 3). It is “middleware” in that it stands between the database backend and the user. It has a command line interface and includes a query tool (EQT) that is able to present data in a graphical manner, including displaying binary and n-ary trees. Emdros includes an application programming interface (API) that can be used to connect it to a web-based application or a standalone query tool implemented by the user. Scripting languages can access an Emdros database, such as Perl, Python, Ruby, Java, C#, or PHP.

Why would Emdros interest a linguist? Let’s answer that question with another one: have you ever designed and implemented a database using one of the standard database software (Oracle, Access, PostgreSQL, etc.)? These programs are flexible and powerful, but are not set up for linguistic data such as text corpora. You must wrestle with the design without much help in the way of models. After you have invested a large amount of time and money in building the database, you then come up against built-in limitations caused by your own ignorance and lack of experience in the arcana of database design. Even with a well-designed database, there is the question of figuring out how to query the data in a manner that actually extracts what you expect. Would it not be wonderful if all that design work had already been done by an expert, and there was already a query language exactly suited to your data? You could hire an expert for hundreds of dollars per hour and pour your precious budget down that hole. Or you could use Emdros.

2. WHAT IS EMDROS? Doedens (1994) describes a database for text based upon the concept of “monads.” Monads are simple integers, providing the unchanging text order at whatever granularity of text segmentation the user chooses. Monads could represent individual phonemes, for example, or they could represent paragraphs. It all depends upon the specific application the user desires. A monad may have “features” associated with it, typically linguistic analysis, but in fact any kind of arbitrary notation is possible. The “Monad-dot-Feature” (MdF) provides an elegant abstract mathematical model of text that can be implemented in software simply and cleanly.

In 2001, Ulrik Sandborg-Petersen released his implementation of the MdF model, extending it slightly; hence the name EMDROS: Engine for MdF Database Retrieval, Organization, and Storage. The user interacts with Emdros via the MQL query language.

1 Ulrik Sandborg-Petersen, PhD, is a Danish computational linguist and an assistant professor at...
There are four concepts to an Emdros database. The figure below, taken from the Emdros website (http://emdros.org/emdf.html), illustrates a simple Emdros database using the Emdros logo as the text.

An Emdros database is a set of sequential *monads* ranging from 1 to infinity. *Objects* are sets of monads, not necessarily sequential. An object is a member of an *object type* (e.g., noun, phrase, or clause), and has *features*. Features are attributes or values associated with an object and usually contain the linguistic information about that object. In the example, there are two objects, *Letter* and *Name*. The object type *Letter* uses only one monad. It has only one feature, *Surface*, which contains the value of the letter. The *Name* object type on the other hand, encompasses six monads since there is only one *Name* object.

Each object has an identification number, called an *id_d*, that uniquely points to an object instance. An id_d is not the same as a monad, although it might appear to be so in this example. The id_d of the only *Name* object is “7,” and the individual letters each have an id_d from 1 to 6. It is only coincidence that the id_d and the monad number of the six *Letter* objects are the same.

2. SCHEMING IN EMDROS. Emdros’s great advantage is its flexibility. It does not force the user into any particular data model or theory of language. But with freedom comes responsibility: the user must provide Emdros with a model or *schema* of the data. The schema defines what the object types are, their associated features, and the data types and allowed values of those features. There is often more than one way to model the data, and the user must decide which model is most appropriate to the use to which the database will be put. How does one decide what objects and features to create and in what relationship to each other? First, of course, are the logical relationships as the user understands them. Second, how does the user desire to *search* or *query* the data? The Emdros schema defines the objects that can be searched and the relationships between those objects. If a database has “verb” objects and “subject” objects, then patterns or values of the associated features of these objects can be placed in queries.

Aalborg University, Denmark. He has worked on Emdros since 1996. His PhD thesis was on Emdros and its application to the works of a Danish playwright. More information about him and his projects can be found at his website: http://ulrikp.org.
3. CASE STUDY. A concrete example of how an Emdros database is constructed will help the reader to see how the process actually works. The example is taken from the work of the J. Alan Groves Center for Advanced Biblical Research. For the past five years we have been creating a treebank of syntactic analysis of the Hebrew Bible (Old Testament). The data is generated by a rule-driven computer parser in an effort to place Hebrew syntax on a firm modern basis according to the best practices of computational linguistics.

The parser outputs syntax trees in an XML format, illustrated in Figure 2 below:

```
CREATE OBJECT TYPE
WITH SINGLE MONAD OBJECTS
HAVING UNIQUE FIRST MONADS
/* the terminal morpheme */
[Word
  parent: id_d;
  id: string;
  morphId: string;
  cat: terminal_cat_t;
  utf8: string;
  michigan: string;
  utf8lemma: string;
  lemma: string;
  strongs: string;
  gloss: string;
]
```

**Figure 2:** XML trees from Ruth 1:1.

We map the XML elements to Emdros object types. For example, the top-level linguistic object is *Tree*, which contains two additional types, a *Node* for nonterminal nodes of the tree, and a *Word* object type for terminal nodes. The XML attributes for `<N>` and `<W>` elements become the features for those object types. To create the object types, the following MQL commands are given to Emdros:
CREATE OBJECT TYPE
WITH MULTIPLE RANGE OBJECTS
/* the non-terminal Node */
[Node
        parent : id_d;
        id : string;
        cat : nonterminal_cat_t;
        start : integer;
        end : integer;
        rule : string;
        head : boolean_t;
    ]

CREATE OBJECT TYPE
WITH MULTIPLE RANGE OBJECTS
HAVING UNIQUE FIRST MONADS
/* the high-level Tree */
[Tree
        id : string;
        alt : alt_enum_t;
    ]

This is only an extract of the complete schema. The datatype of a feature (string, integer, etc.) affects the efficiency of storage and retrieval, as well as providing a way of protecting the data from incorrect input.

Once the schema has been given to Emdros, the next step is to populate the database with actual instances of these object types, including their features and values. Since the treebank contains more than 100,000 trees, doing this by hand is impractical and prone to error. The solution in this case was to write a Python script that has facilities to handle XML. The script outputs MQL statements creating the objects:

CREATE OBJECTS WITH OBJECT TYPE [Node]
CREATE OBJECT FROM MONADS= { 1160001 } WITH ID_D=3770003
    id:="id2";
    rule:="Cj2Cjp";
    start:=0;
    head:=false;
    end:=0;
    parent:=3770002;
    cat:=cjp;
]
...

CREATE OBJECTS WITH OBJECT TYPE [Word]
CREATE OBJECT FROM MONADS= { 1160001 } WITH ID_D=3770004
    utf8Lemma:="\xd7\x95\xd6\xb0";
    morph:="Pc";
Since trees are inherently recursive, the script takes advantage of this fact to correctly create parent nodes from the terminal nodes. The resulting MQL statements are then input into Emdros, and the database is now created.

The purpose of all this work is, of course, to be able to search and query the database. While this can be done from the command line, the Emdros package includes an Emdros Query Tool (EQT), which provides not only a graphical interface to the data, but also an example of how users could provide their own tool, using EQT as a model. Figure 3 shows

---

2 The source code for EQT, written in C++, is provided, along with the source code to the entire system. There are “bindings,” which allow one to write a query—or any other—tool using one’s
the results of a query of this database:

The window on the right shows the schema of the database and hence what can be searched. In the top window is a query. A plain English translation of the query would be:

Search the first 1,000 monads (which represent “words” or “morphemes”), and return the trees where the top node has a “category” feature with the value of “CL” (clause). This node consists of two and only two nodes with no object intervening (indicated by the exclamation mark), the first node having the category value of “P” (predicate), and the second node having the category of “S” (subject).

In “ordinary” linguistic terms:

Find all verbless (nominal) clauses consisting of two and only two constituents, where the predicate is clause-initial and the subject is clause-final.

The bottom window displays the fourth and last “hit,” which is Genesis 2:12, beginning with the fifth word in the verse. The display shows a tree, reading right to left. Emdros understands Unicode and in this case, UTF-8 encoding. EQT can be configured to display search results in a number of different ways, presenting various types of information.

4. CONCLUSIONS. Emdros has a tremendous advantage over the other options available for linguistic databases in that the query language adapts to the user’s conception of the data. The user does not have to struggle with mapping or “translating” a query to the way the user thinks about the data. MQL adapts itself, so that the query is a natural expression of the question. The reduction of error and uncertainty about the correctness of the query is itself alone worth the effort of implementing the database in Emdros.

And that very advantage is the source of the cost3 of using Emdros: learning how to create schemas and writing scripts to convert data from its present form. In this description, I have left out many details that must be dealt with in each unique situation. Fortunately, Professor Sandborg-Petersen is very responsive to user requests, and actively maintains and continues to extend and improve the software. The software is hosted on SourceForge4 and a user forum5 where both the newcomer and the veteran user can find help and education.

One final advantage remains to be noted. Emdros is available for all the major computing platforms: Linux, OS X (Mac), and Windows. Emdros can be run as a server daemon on a network, allowing users to access Emdros databases with only an EQT-type client or even at a remote command line. Emdros databases and MQL files can be shared across preferred programming language, e. g., Perl, Python, Ruby, etc.

3 Emdros’s developer also provides commercial licensing and consultation services, for those with the means.

4 http://sourceforge.net/projects/emdros/

5http://sourceforge.net/forum/forum.php?forum_id=116433
platforms without any modification or conversion. This allows great freedom and flexibility in collaboration with others.

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pros:</strong></td>
<td>- a search engine and query language optimized for annotated text</td>
</tr>
<tr>
<td></td>
<td>- scalable to data containing tens of millions of words</td>
</tr>
<tr>
<td></td>
<td>- query language can be adapted to any linguistic model</td>
</tr>
<tr>
<td></td>
<td>- excellent maintainer and user community support</td>
</tr>
<tr>
<td></td>
<td>- import tools available for standard data formats (Penn treebank, SIL standard format, TigerXML, and others)</td>
</tr>
<tr>
<td><strong>Cons:</strong></td>
<td>- significant learning curve</td>
</tr>
<tr>
<td></td>
<td>- programming skills are needed to populate databases for which there is no standard importer</td>
</tr>
<tr>
<td></td>
<td>- primitive user query and report tools (although they are improving)</td>
</tr>
<tr>
<td><strong>Primary function:</strong></td>
<td>Database engine for analyzed or annotated text</td>
</tr>
<tr>
<td><strong>Platforms:</strong></td>
<td>Linux, OS X, Windows, Solaris, *BSD</td>
</tr>
<tr>
<td><strong>Open Source?:</strong></td>
<td>Yes, licensed under GPL, v. 2. Source code and binaries downloadable from <a href="http://emdros.org/download.html">http://emdros.org/download.html</a></td>
</tr>
<tr>
<td><strong>Proprietary?:</strong></td>
<td>Dual licensed: the user can use Emdros under the GPL license, or negotiate a commercial license with the developer.</td>
</tr>
<tr>
<td><strong>Reviewed version:</strong></td>
<td>Emdros version 3.0.2.pre08</td>
</tr>
<tr>
<td><strong>Application size:</strong></td>
<td>176 MB for all programs, source code, and documentation. The installation of the rpm binary package requires 12MB.</td>
</tr>
<tr>
<td><strong>Documentation:</strong></td>
<td><a href="http://emdros.org/docs.html">http://emdros.org/docs.html</a></td>
</tr>
</tbody>
</table>
REFERENCES
