A Lexicon Module for a Grammar Development Environment

Ann Copestake[*], **Fabre Lambeau**[*], **Benjamin Waldron**[*],
Francis Bond[‡], **Dan Flickinger**[†], **Stephan Oepen**[†⋆]

[*]University of Cambridge Computer Laboratory
JJ Thomson Avenue, Cambridge CB3 0FD (UK)
{aac10,faml2,bmw20}@cl.cam.ac.uk

[†]CSLI, Ventura Hall, Stanford University
Stanford, CA 94305-4115 (USA)
{danf,oe}@csli.stanford.edu

[‡]NTT Communication Science Labs
2-4, Hikaridai, Seika-cho, Soraku-gun
Kyoto 619-0237 (Japan)
bond@cslab.kecl.ntt.co.jp

[⋆]Institutt for Lingvistiske Fag
Universitetet i Oslo
0317 Oslo (Norway)

Abstract

Past approaches to developing an effective lexicon component in a grammar development environment have suffered from a number of usability and efficiency issues. We present a lexical database module currently in use by a number of grammar development projects. The database module presented addresses issues which have caused problems in the past and the power of a database architecture provides a number of practical advantages as well as a solid framework for future extension.

1. Introduction

This paper concerns the use of a relational database as a component of a grammar development environment (GDE). The concept of a lexical database is of course very well-known, with WordNet being the primary example. But there is little reported work on the design of the base lexical component within a GDE: that is, on the component which associates stems with syntactic and semantic information. (There is, of course, considerable work on lexical representation within morphology, but this raises rather different issues.) One reason for this is that speed of lexical lookup is not a primary consideration in most systems, so while developing efficient parsing algorithms is a core NLP problem, efficiency of lexical access is not. But much of the work on parsing efficiency has been done with relatively small grammars with tiny lexicons. Grammar engineering aspects, such as lexicon maintenance, have been neglected in the published literature.

The work reported here addresses these practical issues. It concerns the use of a standard relational database system as an integrated part of a GDE – specifically the LKB system (Copestake, 2002) - although we believe that most of the issues we raise are applicable to other processing environments. We base the description that follows on the ERG – the LinGO English Resource Grammar (Copestake and Flickinger, 2000) – although the lexical module is also in use with grammars of other languages such as Japanese[1] and Norwegian[2].

2. The Past Approach

Previous versions of the LKB relied on purpose-built lexical database code. A lexicon, as seen by the grammar writer, consisted of a series of text files written in the same syntax as the rest of the grammar. For example in Figure 1 the identifier is `bombard_v1`; the syntactic/semantic type is `v_np_trans_le`, which would be expanded into a large typed feature structure (TFS) when the entry was used in the system; the orthography is `"bombard"`; and the semantic predicate is `"_bombard_v_rel"`. These lexicon files were converted to a special-purpose database cache by the LKB system when loading a grammar, from which individual lexical entries could later be retrieved for expansion into a TFS.

```
bombard_v1 := v_np_trans_le &
 [ STEM < "bombard" >,
   SYNSEM.LKEYS.KEYREL.PRED
                          "_bombard_v_rel" ].
```

Figure 1: A lexical entry in textual format

2.1. Issues

This approach worked adequately in terms of processing efficiency, even with large lexicons: lexical access is not a significant fraction of parse times. However it had a number of severe disadvantages. A lexicon encoded as a large text file is very difficult to maintain. Lexical lookup was restricted to values which had been explicitly indexed (orthography and lexical identifier), meaning that, for instance, a grammar developer could not easily retrieve and modify all entries of a given type. Loading a large lexicon could be time-consuming, but the cache mechanism was not robust to changes in the lexicon or code. The only place to store information was within the feature structures themselves or as a comment in the text file; this did not allow us to keep track of bookkeeping information. Checking could only be done when explicitly invoked: checking the entire lexicon was too expensive to do every time a grammar was loaded.

One could attempt to address some of these issues by outputting the text files automatically from an offline database. We experimented with this approach for some time, but it proved too cumbersome for the grammar developers to make use of it systematically.

[1]The JACY Grammar (Siegel and Bender, 2002).

[2]NorSource (www.ling.hf.ntnu.no/forskning/norsource)

lexid	`bombard_v1`
type	`v_np_trans_le`
orth	`"bombard"`
sempred	`"_bombard_v_rel"`
source	`ERG`
userid	`danf`
timestamp	`2003-11-01 00:00:00+00`
lang	`EN`
confidence	`1.0`
comments	`<UNINSTANTIATED>`
...	

Figure 2: Part of a sample database record

3. A Database Lexical Module

Our new version of the LKB lexicon module makes direct use of PostgreSQL (The PostgreSQL Global Development Group, 2003). PostgreSQL is an object-relational database management system which we chose primarily because of its open source status. Our lexicons are maintained within PostgreSQL databases and these databases are directly accessed by the LKB during parsing, generation and so on. This approach allows the use of all the standard database queries and update routines, which considerably enhances lexicon management as well as processing speed. Caching as used in the past is no longer required and we are able to make use of a sophisticated range of filters (as explained below) to change the accessibility of lexical entries for parsing and generation. The new lexical module requires 10-15 seconds[3] to load the ERG grammar (11.9K lexical entries), in contrast to the 45 seconds needed in the past; the JACY grammar (35.5K lexical entries) now requires 10-20 seconds, reduced from 2 minutes. Likewise, checking the full lexicon now takes 20 seconds for the ERG, reduced from 35 seconds; JACY now requires 70 seconds reduced from 135 seconds.

The relational database encoding assumes that the lexical information for simplex words is a tuple of grammatical ‘slots’ from which the full TFS of the entry may be derived (in a similar manner, the textual representation specified a set of TFS path values which taken together allowed one to construct the full TFS). Multiword expressions, such as idioms, are more complex (Copestake et al., 2002) but can also be encoded (§ 4.3.). For bookkeeping purposes, we also store information about the source of the entry, the date it was created, the lexicographer, confidence scores and so on. Using this bookkeeping data we are able to define a particular view of the lexicon by means of a filter. A filter specifies that we are interested only in a subset of those lexical entries stored in the full database – e.g. we may only want to use entries from a particular source with a particular confidence threshold, or to omit entries pertaining to non-standard dialects. Such filters can be encoded as follows:

```
source = 'corpusA' AND confidence > 0.5
dialect = 'standard'
```

[3]All timings quoted relate to typical 2.2 GHz CPU machine.

4. The Database Structure

We store the lexicon within a PostgreSQL relational database. The database contains a history for each lexical entry, and has a structure able to accommodate multiple users and varying ‘views’ of the data; a particular subset of the lexicon is chosen at load time by applying a filter on the bookkeeping fields, and it is this subset (‘view’) which will later be directly queried by the LKB during lexical lookup. Such views are cached internally to boost lexical lookup speed.

At lexicon load time the LKB establishes a connection to a PostgreSQL database and specifies the required view of the database. The PostgreSQL server may run locally, or alternatively can be made available from a remote machine. In order to accommodate multiple concurrent user logins with multiple environments (and to accommodate simultaneous edits by multiple lexicographers) we utilise a public schema and multiple private schemas. The public schema contains the bulk of the lexical database (LexDB) in addition to other data and functionality shared by all users, whilst each user login possesses a private schema used to store that user’s working edits of the LexDB, their current view of the LexDB and other data specific to that user.

4.1. Public Schema

Lexical entry revisions are stored in the database with associated bookkeeping fields in addition to the grammatical fields which provide the information necessary to reconstruct the TFS of the lexical entry (see Figure 2 for a sample database record; the first four fields are the grammatical fields which provide the information which in the past was stored in the textual lexicon files). Each grammatical field maps to a particular TFS path; for example an entry in the predicate field may map to a TFS component equivalent in the textual notation to:

```
SYNSEM.LKEYS.KEYREL.PRED "_bombard_v_rel"
```

The field entries are not restricted to mapping to atomic values; for example, orthography will map to a list which may contain multiple elements (e.g. *ad hoc*). The mappings from fields to TFS substructures are themselves defined in a database table; as such they may be edited as the grammar evolves, and different ‘modes’ may be used to define different sets of mappings (allowing distinct grammar encodings to utilise the same lexical module).

The bookkeeping fields include userid and timestamp (used to specify a unique revision of a lexical entry), documentation fields such as comments and exemplars, and fields such as confidence, source, language, country, dialect, domain, genre and so forth. This bookkeeping data is not available when using entries from textual lexicon files (human-readable comments are possible but cannot be processed in any effective way).

The grammatical and bookkeeping fields are an integral part of the LexDB. Additional lexical data is made available when we link entries to external database resources (see §6.). This at the time of writing includes a semantics database and a database of idioms.

The database is queried directly during lexical lookup and other tasks. The database code (that is, the set of

SQL queries) involved is defined in the database itself (via database functions and using embedded SQL code in cases where database functions do not suffice). The idea is to ensure the lexical module is distinct from any particular 'client' (such as the LKB). Alternative clients may include those performing editing tasks (see §5.), those for querying/displaying (e.g. a web interface), as well as in future alternative grammar processing tools such as the PET system (Callmeier, 2000).

4.2. Private Schemas

The private schemas allow us to provide the multi-user functionality. For example, whilst editing the LexDB changes should be invisible to other users but should be available as part of that user's view of the lexicon. So personal edits are stored in the private schema (and effectively added to the LexDB for all operations of the current user). The database filter is also be specific to each user, as is their current view of the LexDB.

4.3. Multi-Word Entries

Multi-word entries (MWEs) come in two classes. Members of the class of non-decomposable MWEs, for which an exemplar is *ad hoc*, are treated simply as simplex lexical entries. Members of the class of decomposable MWEs, for which *spill the beans* and its variants provide an exemplar, are treated via templates and allow for a certain degree of variation. See (Copestake et al., 2002) for a discussion of issues and grammatical representation.

The lexical database stores multi-word entries of the second class in separate tables to the simplex entries. Each instance is associated with a type for the template as a whole, and with specifications for each template slot (corresponding to *spill* and *beans* in the above example). These slots require that idiomatic simplex forms be added to the main lexicon. These idiomatic forms are derived from basic simplex forms (e.g. idiomatic *spill* differs from basic *spill* only in its semantics). Such idiomatic forms would ideally derive in the grammar from the base forms via a default inheritance mechanism, however for current purposes they occupy a separate table in the lexical database, which is used to generate idiomatic simplex forms which (conceptually) are added to the original set of simplex forms in the database.

The grammar-specific data outlined above may be augmented by linking such entries to the multilingual database of idioms described in (Villavicencio et al., 2004).

5. Tools

5.1. Emacs Interface

The lexical module is designed in such a way that various clients may interact with the database server. Here we discuss an interface to a text editor (Emacs).

Easy and effective editing of lexical entries is vital if the database module is to be useful for grammar development. The lexical module as it existed in the past utilised text files of the same format as the rest of the grammar. As such they could be edited with a text editor (generally Emacs) in the same way as the other files. Commercial database packages tend to provide effective mechanisms for editing, viewing and presenting the contents of a database. Unfortunately PostgreSQL lacks such user-friendly interfaces; the various interfaces that exist are not suited to our task. We have developed a custom Emacs interface to the lexical module. Such an interface is important because the lexicon is generally edited at the same time as the type files on which it depends; it would be inconvenient to require the use of multiple editor applications for what is essentially a single task. The main grammar developers/lexicographers are already familiar with Emacs.

The Emacs database interface (written in Emacs Lisp) allows existing entries to be edited and new entries to be added. Entries in the database may be browsed and cross-indexed, tab completion may aid the lexicographer by presenting possible completions for the current field based on the contents of other lexical records, and so on. New revision entries produced in this way reside initially in the user's private schema, from where they may be committed to the public table when the user is ready. Adding a new revision entry achieves one of three purposes; alter an existing lexical entry, add a new entry, or remove an existing entry (the revision history is not lost, the head revision merely specifies that the lexical entry is not to be used).

5.2. CVS

The LinGO grammar development projects generally use a version control system, such as CVS, where the grammar is specified as a set of human-editable text files. The old lexical module used text files in exactly this manner. Lexical entries in the database module are stored as part of a database system and are not immediately available in such a format. Hence we share the lexicon on CVS via a set of database dump files containing the entry- and grammar-specific data held in the lexical database (one text file for each such table in the public schema of the database). These text dumps are effectively human-readable (and editable), but the idea is that they are simply a means of storing and distributing the database contents. Forks in lexicon development are automatically merged on CVS check in – there are no conflicts as each revision is unique to a particular user (and timestamp); equally, when checking out an updated lexicon from CVS the new revisions are simply merged into the database resident on the database server.

6. Additional Database Resources

By moving lexicon development into a lexical database module we gain the opportunity to take advantage of additional database resources. We currently augment the LexDB by linking to a database which holds detailed lexical semantics (effectively Minimum Recursion Semantics definitions (Copestake et al., 1999)). Among other purposes, this acts as a data source when performing generation; we avoid the necessity of a highly resource-intensive batch process prior to performing generation on the grammar.

An additional external resource currently in use is a database of idioms (Villavicencio et al., 2004). This links idiomatic lexical entries (both simplex and the more complex MWE forms) to extensive data on the meaning (and foreign language equivalents) of the idioms.

```
colon-advp := adv-p-lex-1 &
 [SYNSEM.LOCAL
  [CAT.HEAD [PTYPE comma],
            CONT.RELS <! [PRED 'wa] !>],
  ORTH <! ":" !> ].
```

Figure 3: A non-standard lexical entry in textual format

7. Issues

The database module addresses the major maintenance and efficiency issues outlined in §2. By taking advantage of the bookkeeping fields and custom tools (§5.) effective maintenance of large lexica is facilitated. Efficiency issues such as load time are addressed due to the speed of the database access operations, and by the opportunity to do away with the old caching mechanism – caching of lexicon views can still be worthwhile with a large lexicon, but now the times involved are less by an order of magnitude than in the past, and such caching is robust to edits to the lexicon.

But although developing this module was originally predicted to be a reasonably simple task, a number of issues have arisen. For most users, the requirement to run a PostgreSQL database server makes it considerably more difficult to install the LKB; because of this, the old lexicon code is still supported. Supporting multiple operating systems is more complex; we currently only support the LKB with PostgreSQL under Linux. Source control becomes considerably more complex. The distinction between bookkeeping and other information is not completely clear; for instance, dialect information has clear linguistic significance but may not need to be encoded in the feature structures.

The requirement to convert the lexicon to the standardised tuple required for the database is non-trivial for many grammars, especially for closed class words. Our implementation thus allows for a hybrid approach, where some entries are stored in the database while others are represented in text files as a temporary measure while the lexicon is standardised. This standardization may generally be achieved by moving certain definitions from the lexicon into the type system; for example the entry shown in Figure 3 (taken from the JACY grammar) can be standardized by moving the contents of *SYNSEM.LOCAL* into the type system, because the remaining parts of the definition fit into the 'type' and 'orthography' database fields respectively. This hybrid approach utilises a hierarchy of lexicons in which the lexical database forms the main lexicon while additional lexical sources form sublexica. Since grammar developers are often adding to or modifying the lexicon at the same time as the other grammar files, having to access separate database interface functionality can be a nuisance; hence the hybrid approach is thus also useful for quick experiments with the lexicon when extending the grammar.

8. Conclusion

This paper has presented a lexical module utilising the benefits of a database architecture. Such a module is able to address issues such as lexicon maintenance, lookup and efficiency which have proven problematic in the past. Although we have based our illustration on a specific GDE and a specific grammar, the system is sufficiently modular to extend to alternative GDEs and is currently in use as a component in the development of multiple large grammars.

Both the LKB system and the ERG grammar are open source and are downloadable from lingo.stanford.edu. (A longer version of this paper can be found at www.cl.cam.ac.uk/users/bmw20.)

9. Acknowledgements

This document was generated partly in the context of the DeepThought project, funded under the Thematic Programme User-friendly Information Society of the 5th Framework Programme of the European Community (Contract No IST-2001-37836).

10. References

Callmeier, Ulrich, 2000. PET. A Platform for Experimentation with Efficient HPSG Processing Techniques. *Journal of Natural Language Engineering*, 6(1):99–108.

Copestake, Ann, 2002. *Implementing Typed Feature Structure Grammars*. CSLI Publications, Stanford, USA.

Copestake, Ann and Dan Flickinger, 2000. An open-source grammar development environment and broad-coverage English grammar using HPSG. In *Proceedings of the Second conference on Language Resources and Evaluation (LREC-2000)*. Athens, Greece.

Copestake, Ann, Dan Flickinger, Ivan Sag, and Carl Pollard, 1999. Minimal Recursion Semantics: An introduction. Unpublished Manuscript (www.cl.cam.ac.uk/users/aac10/papers/newmrs.pdf).

Copestake, Ann, Fabre Lambeau, Aline Villavicencio, Francis Bond, Timothy Baldwin, Ivan Sag, and Dan Flickinger, 2002. Multiword Expressions: Linguistic Precision and Reusability. In *Proceedings of the 4th International Conference On Language Resources and Evaluation (LREC-2002)*. Las Palmas, Canary Islands.

Siegel, Melanie and Emily Bender, 2002. Efficient Deep Processing of Japanese. In *Proceedings of the 3rd Workshop on Asian Language Resources and International Standardization. COLING-02 Post-Conference Workshop*. Kyoto, Japan.

The PostgreSQL Global Development Group, 2003. PostgreSQL 7.4.1 Documentation. (www.postgresql.org).

Villavicencio, Aline, Timothy Baldwin, and Benjamin Waldron, 2004. A Multilingual Database of Idioms. In *Proceedings of the 4th International Conference On Language Resources and Evaluation (LREC-2004)*. Lisbon, Portugal.