Hydra: A Software System for Wordnet

Borislav Rizov
Department of Computational Linguistics
Institute for Bulgarian
Bulgarian Academy of Sciencies
Sofia, Bulgaria
`boby@dcl.bas.bg`

Abstract

This paper presents an overview of the software for wordnet processing Hydra. The system has fully-fledged GUI and API, both working with powerful modal query language. Hydra has been used for the development of the Bulgarian WordNet for the last 7 years and recently was improved, became open source and is distributed as part of the Meta-Share platform.

1 Introduction

During the development of Bulgarian WordNet (BulNet (Koeva et al., 2004)) at IBL[1] the need for a convenient and powerful tool for creating and processing wordnet arose. Multiple applications of wordnet in various computational linguistics tasks suggested definition and implementation of API (Application Programing Interface) to work with Wordnet as well. The presented system, Hydra, solved these problems, and provided additional benefits such as abstract mathematical query language, concurrent user access, undo / redo of user operations, synchronization between languages. As part of the project CESAR[2] Hydra was improved, the range of the supported linguistic databases that it can work with was increased, configurability and use by end users was greatly facilitated. Hydra's code was opened and is currently used by several teams working with wordnets for various languages like Croatian and Romanian.

2 Overview

Hydra is a system for dealing with lexical-semantic networks such as wordnet. It is open source under the GPL v3 license and it is available at: `http://dcl.bas.bg/en/hydra.html`. The program has a convenient and rich user interface. Hydra provides an API for access to the semantic networks of this type, which provides an abstract and easy access to such linguistic databases. It was used in the last several years for the development of BulNet. The relational model that Hydra uses is generic enough and allowed the archive of the Department of Bulgarian Dialectology and Linguistic Geography at IBL[3] to be imported and the user interface and API of Hydra used for its processing.

Hydra supports all the operations necessary for the creation of electronic linguistic databases similar to wordnet (definable in terms of a relational structure). The main features include editing of existing synsets and relations adding, editing and deleting a synonym set, reverting a single action (undo) or group of actions (cancel), returning a canceled operation (redo). The second type of features includes two operations (i) creation of synsets and relations which do not have analog in another Wordnet (e.g. language-specific concepts), and (ii) cloning synsets, an operation where synset is copied from one language wordnet to another.

Hydra is implemented in Python[4], using the platform independent GUI library Tkinter. The data is managed by a MySQL server, which remains hidden from the end users after the initial installation. The system has been successfully used on Windows and Linux.

Hydra has the following important features:

- The program allows users to edit or query any number of wordnets simultaneously. Individual wordnets can be synchronized, allowing

[1]`http://dcl.bas.bg/`
[2]`http://cesar.nytud.hu/`
[3]`http://ibl.bas.bg/en/departments_en4.htm`
[4]`http://python.org/`

simultaneous visualization of the equivalent synsets in different languages.

- Allows concurrent access by multiple users.
- The changes in the database are available to all users right after they are made.
- Powerful modal language for searching in linguistic data (Rizov, 2008), as well as an user interface with variety of predefined query utilities:
  - simple queries on words and combinations of words
  - search with regular expressions using MySQL syntax
  - search formulas - complex searches based on the Modal Language for WordNet.
- Enables checks for completeness and consistency, some of which are built into the program.

3 Wordnet as a Relational Structure

This paper does not aim to describe the properties and applications of wordnet. Let us just recall the main features to focus on the proposed solution. Words of the language are divided into synonym sets (synsets) and their relationships expressed in relations such as hyperonymy, antonymy, etc (semantic, morpho-semantic and other). (Vossen, 2004) The modal approach to logical representation of this formalism in Hydra suggests that wordnet is encoded as a relational structure: a set of objects and a set of binary relations between them. Consider the data in Wordnet. We have synsets provided with:

1. Identifier that is common to the equivalent meanings (synsets) in different languages (ili)
2. part-of-speech (pos)
3. encyclopedic definition (definition)

These data will be designated as *single type* because they have just one instance in one synset. We also have those of *multiple type* such as usage examples, the synsets notes (snotes) and others. Synsets comprise several words. They have, in the Bulgarian WordNet, the form of the word/compound word (word), basic form (lemma), and a unique number to identify the word sense (sense). These are the data of *single type*. The members of a synset often are provided with notes that are of *multiple type* – we may have any number of them. The following convention is adopted for encoding the data as a relational structure. Objects contain all *single type* data. Any object of multiple type is a separate object and its belonging to the original object is expressed by a relation. Thus the following 3 types of objects are defined (we call them *linguistic units*). **Synset** contains the data: pos, ili, definition and other single type data. **Literal** represents a word in a synset and contains the data: word, form, sense. Membership of a literal to synset is expressed by the relation *literal*, so every literal in a synset is associated with a single synset with this relation. The third type of object is formally called **Note** and presents text information such as examples and notes. Several provisional relations such as *literal* are responsible to assign objects to their 'owners'. For example, usage examples are associated with synsets with the relation *usage*. An important assumption is that each object is associated to exactly one synset. Each synset is associated to itself, each literal to the synset it belongs to and each note is 'part of' exactly one synset or literal and thus inherits its synset association. This association is not explicit but it is important and is true in the other wordnet representations. It allows to synchronize linguistic units from different languages. This is achieved by synchronizing their synsets. Here it is appropriate to mention that synsets in different languages, which have the same meaning, are connected by the relation *ili*.

4 Modal Language for Wordnet

The main task of the modal language is to provide a clear formalism for queries with sufficient expressive power with which to address the major problems in dealing with Wordnet. This includes search, validation, synchronization of languages, etc. This modal language is easy to learn and use for the average user and does not require specialized knowledge of databases and programming which is common for other approaches. Another advantage of this abstraction is that it hides the data presentation from the user and allows its various implementations and modifications. Thus, it is extremely easy to add new relations or data (single type) in the already defined types.

Modal language in Hydra is based on that given in (Koeva et al., 2004).

4.1 Syntax and Semantics

Detailed syntax and semantics of The Modal Language for Wordnet is given in (Rizov, 2008). We will present how the syntax looks in Hydra and also its informal interpretation. Note that for a given formula the system returns all objects that are a model for it (the formula is true in them). Also, we will use the term query, together with a formula which is natural in the context of the system Hydra.

Atomic Queries:

- Each object in the database has a primary key and it is a nominal (constant) in our language. They are natural numbers divided into 3 disjoint sets, and thus their type is identifiable. Examples: 1 – Literal, 12111003 – Note, 1231100311 – The synset nominals are encoded to be portable and depend only on ili, pos and the language of the synsets they denote.
- constants $s – all synsets, $l – all literals, and $n – all text objects (of type Note) at the linguistic database.
- constants for fields of objects, $type('value')$, such as word('person'). Returns items that have a field $type$ with value $value$. To use a regular expression, add # before the first quote – word(#'c[au]t').

Queries:

- Atomic queries are queries.

  Let q and r be queries (formulae), then the following queries are true in the objects where:
- !q – q is not true;
- q & r – q and r are true;
- q | r – q is true or r is true;
- q => r – q is not true or r is true;
- q <=> r – q and r have the same thruth value.

  Let also R be a relation:
- In x the query <R> q is true if there is an object y, xRy and q is true in y. In other words, find those objects, for whose neighbours by the relation R the query q is true. For example, to find hypernyms of synset with number 10140069453, we need the query <∼hypernym>10140069453 (∼ R is the reversed relation of R) or <hyponym>10140069453.
- <R,n> q is true in the object x iff $|\{y \mid xRy \wedge y \Vdash \varphi\}| > n$. So to find the synsets with more than one hypernym we can use the query <hypernym, 1>$s
- In x is true <R,n:m> q iff $|\{y \mid xRy \wedge y \Vdash \varphi\}|\, m > n\, |\{y \mid xRy\}|$

4.2 Example Queries

Here are some example queries and how they are expressed in the defined language:

- Find all literals that have word with value *game*: **word('game')**, then all of its meanings (synsets): **<literal>word('game')** and their meanings in bulgarian (bg): **<ili><literal>word('game')&lang('bg')**
- **ili('eng-30-01815628-v')** - returns the synset with the ILI eng-30-01815628-v in every wordnet in the wordnet database in which it is found
- **<snote>$n** - retrieves all the synsets that have at least one Snote
- **<literal><lnote>note('pl. t.')** – synsets that contain literals having an lnote pl. t. D. Searching in synset-to-synset relations
- **<hypernym>ili('eng-30-02396716-v')** – matches all the synsets that share a hypernym

5 Graphical User Interface

The user interface consists of a search window and a window with dictionaries. The search window provides the entry point to the data in the linguistic database. It also provides for opening dictionaries for the languages. A very useful innovation is the loading of results from external sources. File / Open menu command loads the file, assuming that the first word of each non-empty line is an identifier (nominal) of an object in the database. The same result is achieved by entering the path to the file in the search box prefixed with 'file:', e.g. 'file:/home/boby/biology.txt'. This functionality provides an easy way for using results from external scripts (for example, those who use API of Hydra). It is very important for some data extractions that cannot be expressed with the modal language, such as some transitive closures of relations. For example, we can find all the hyponyms of a given synset (not only immediate one) with this simple script:

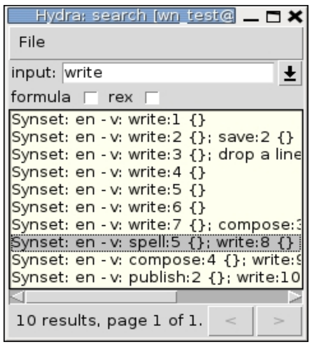
```
from wordnet import wn

def hyponyms(synset):
    for h in wn.relations['hyponym'].neighbours(synset):
        print h.ID()
        hyponyms(h)

hyponyms(wn.ling(1231100311))
```

Then we can start it, store the output in file hyponyms.txt and open it in the searcher.

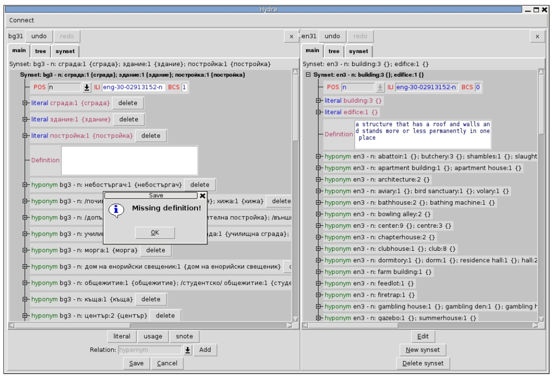
```
 $ python hyponyms.py > hyponyms.txt
```



```
input: write
Synset: en - v: write:1 {}
Synset: en - v: write:2 {}; save:2 {}
Synset: en - v: write:3 {}; drop a line
Synset: en - v: write:4 {}
Synset: en - v: write:5 {}
Synset: en - v: write:6 {}
Synset: en - v: write:7 {}; compose:3
Synset: en - v: spell:5 {}; write:8 {}
Synset: en - v: compose:4 {}; write:9
Synset: en - v: publish:2 {}; write:10
```

Figure 1: Search Window

Each dictionary in the second window contains multiple views for visualization of a synset. The dictionary is tied to a single language and displays the clone (see API) of the current object in this language. Dictionaries can be synchronized according to the users will.



```
Synset: bg3 - n: сграда:1 {сграда}; здание:1 {здание}; постройка:1 {постройка}
Synset: bg3 - n: сграда:1 {сграда}; здание:1 {здание}; постройка:1 {постройка}
  POS n   ILI eng-30-02913152-n   BCS 1
  literal сграда:1 {сграда}
  literal здание:1 {здание}
  literal постройка:1 {постройка}
  Definition
  hyponym bg3 - n: небостъргач:1 {небостъргач}
  hyponym bg3 - n: /почи ...}; хижа:1 {хижа}
  hyponym bg3 - n: /допъ ...ителна постройка}; /външ
  hyponym bg3 - n: учили ...да:1 {училищна сграда};
  hyponym bg3 - n: морга:1 {морга}
  hyponym bg3 - n: дом на енорийски свещеник:1 {дом на енорийски свещеник}
  hyponym bg3 - n: общежитие:1 {общежитие}; /студентско/ общежитие:1 {студе
  hyponym bg3 - n: къща:1 {къща}
  hyponym bg3 - n: център:2 {център}

Save
Missing definition!

Synset: en3 - n: building:3 {}; edifice:1 {}
Synset: en3 - n: building:3 {}; edifice:1 {}
  POS n   ILI eng-30-02913152-n   BCS 0
  literal building:3 {}
  literal edifice:1 {}
  Definition a structure that has a roof and walls an
             d stands more or less permanently in one
              place
  hyponym en3 - n: abattoir:1 {}; butchery:3 {}; shambles:1 {}; slaught
  hyponym en3 - n: apartment building:1 {}; apartment house:1 {}
  hyponym en3 - n: architecture:2 {}
  hyponym en3 - n: aviary:1 {}; bird sanctuary:1 {}; volary:1 {}
  hyponym en3 - n: bathhouse:2 {}; bathing machine:1 {}
  hyponym en3 - n: bowling alley:2 {}
  hyponym en3 - n: center:9 {}; centre:3 {}
  hyponym en3 - n: chapterhouse:2 {}
  hyponym en3 - n: clubhouse:1 {}; club:8 {}
  hyponym en3 - n: dormitory:1 {}; dorm:1 {}; residence hall:1 {}; hall:2
  hyponym en3 - n: farm building:1 {}
  hyponym en3 - n: feedlot:1 {}
  hyponym en3 - n: firetrap:1 {}
  hyponym en3 - n: gambling house:1 {}; gambling den:1 {}; gambling h
  hyponym en3 - n: gazebo:1 {}; summerhouse:1 {}
```

Figure 2: The Dictionaries Window

Editing and adding new linguistic units is done directly in the main view of the dictionary. Data consistency during concurrent access is provided by locking of the edited units and their neighbours. User actions in navigation and editing can be canceled one by one (undo), in groups (cancel) and redone (redo).

A detailed overview of the user interface and other features of the Hydra system is available in the user guide:

`http://dcl.bas.bg/Tools/Hydra/Hydra-UserManual.pdf`

6 Data Representation

When developing a solution how to store and manage the data, the choice fell to relational DBMS and specifically to MySQL. Hydra instances work directly with the MySQL server and take care for consistency of the data during the concurrent access. Modal formulas of the Language for Wordnet are translated directly to SQL queries, each returning those object where the formula is true. The main data types are stored in the tables corresponding to their names: Synset, Literal and Note. The relation pair are in table Rel. Also the table Relation keeps the definitions of the relations in Wordnet. Hydra is designed to work in a very general case(Gamma et al., 1995). The data in an object type is not strictly fixed. Its structure is configured in module descriptor.py. Consider the structure of Synset.

```
class SynsetId(Table):
    table = 'Synset'
    fields = ('id', 'ili', 'pos', 'definition',
      'stamp', 'bcs', 'lang',
      'frequency','domain')
    foreign = {'pos':  POSId,
      'lang': LangId, 'domain': DomainId}
```

In any such definition there are two mandatory fields: table – specifies the name of the database tables and fields – list of fields in that table. As is shown in the example, there is a third field, which is optional, foreign. It specifies the foreign key fields. Values in the dictionary are the descriptors of the tables whose keys are stored in the respective fields. Here, such field is pos, the part of speech of the synset. The 'Synset' table stores only keys from the table 'POS'. Its descriptor is:

```
class POSId(Table):
    table = "POS"
    fields = ('id', 'name')
```

The use of foreign keys has several advantages. Usually their values are small fixed set. This set is easily accessible and its values can be easily modified without affecting other tables in the database. For example, we can change the name of a part of speech, and that will not change any record in the table Synset. However, users will see the new name in the synsets. Another place where the changes need to be specified in the structure of the data is dbfeeder.py, which is responsible for database creation and feed with data.

7 API

The entry point of the Hydra API is the object *wn* in the *wordnet* module. Search by a formula is carried out with its *get* method. The method receives one argument, formula of the modal language, and returns a list of all the objects in which the formula is true. Objects in the result are in three types of objects, which build wordnet – Synset, Literal and Note. They are defined in the *linguistic_units* module.

To get all the synsets from language 'bg':

```
>>> from wordnet import wn
>>> synsets = wordnet.get("lang('bg')")
```

wn.ling constructs an object by its nominal (its ID in the database).

wn.relations is dictionary of the type 'relation name': object of type Relation (defined in module relations.py)

7.1 Objects

The main wordnet object types inherit the class Ling. Here are its main methods.

1. to_string(field=None) – return the string representation of the object. Can be called with an optional field name argument, in which case it returns its string value.

   ```
   >>> literals = wn.get("word('name')")
   >>> print literals[0].to_string()
   name:1 {}
   >>> print literals[0].to_string('word')
   name
   ```

   More convenient way to access the field is:

   ```
   >>> print literals[0]['word']
   name
   ```

2. edit() – turns the object in edit mode
3. from_string(value, field) – when in edit mode, the field receives the value
4. save() – save the changes and turns the object in non-edit mode.

   ```
   >>> print literal['word']
   name
   >>> literal.edit()
   >>> literal.from_string('NAME', 'word')
   >>> literal.save()
   >>> print literal['word']
   NAME
   ```

5. check() – Used for data consistency checks of the object and its relations. The inherited object provides implementations to maintain the invariants in the Wordnet structure. It is used by the user interface. For example, when saving a Synset it is mandatory to have at least one Literal. Literals are checked to have non-empty field word.
6. clonning(lang) – returns the corresponding object in the language lang. If lang is equal to the object language, the object itself is returned, otherwise the synset with the same ili as the synset associated with our object, but in language lang is returned.

Synset
literals() – returns the list of the literals in the synonym set.

7.2 Relations

Another type is that of the relations – Relation. It provides methods to add and remove elements of relation, use the reverse relation etc. Access to objects for each of the relations in the database is provided by the wn.relations dictionary, the values being of type Relation or its inheritants, such as ReverseRelation.

```
>>> relation = wn.relations['hypernym']
>>> relation['name']
u'hypernym'
>>> relation['rname']
u'hyponym'
>>> synset = wn.get("<literal>word('game')")[0]
>>> print relation.neighbours(synset)[0].to_string()
en - n: activity:2 {}
```

The example demonstrated the method *neighbours*, which returns the immediate neighbours of the given linguistic object.

7.3 Applications

The API is used in many products of DCL like the DCL Search Engine[5], Bulgarian WordNet–web access[6] (RESTful webservice) etc. The GUI classes were used for the open source corpora annotation tool Chooser[7] but their use is beyond the scope of this paper.

Acknowledgments

This paper was prepared within the project Integrating New Practices and Knowledge in Un-

[5]http://search.dcl.bas.bg

[6]http://metashare.ibl.bas.bg/

[7]http://dcl.bas.bg/en/Chooser.html. Also available at Meta-Share

dergraduate and Graduate Courses in Computational Linguistics (BG051PO001-3.3.06-0022) implemented with the financial support of the Human Resources Development OP 2007-2013 co-financed by the European Social Fund of the EU. The author takes full responsibility for the content and under no conditions can the conclusions be considered a position of the European Union or the Ministry of Education, Youth and Science of the Republic of Bulgaria.

References

Erich Gamma, R. Helm, R. Johnson and J. Vlissides. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

Svetla Koeva, S. Mihov and T. Tinchev. 2004. *Bulgarian Wordnet - Structure and Validation* volume 7, No. 1-2: 61–78. Journal: Romanian Journal of Information Science and Technology.

Svetla Koeva 2010. *Bulgarian Wordnet - current state, applications and prospects*. Bulgarian-American Dialogues, Prof. M. Drinov Academic Publishing House. Sofia. 120–132

Borislav Rizov. 2008. *Processing Wordnet with Modal Logic*: 93–100. Proceedings of the 6th International Conference on Formal Approaches to South Slavic and Balkan Languages.

Piek Vossen. 2004. *EuroWordNet: A Multilingual Database of Autonomous and Language-Specific Wordnets Connected via an Inter-Lingual Index*. International Journal of Lexicography, 17(1): 161–173, June.