A Tool for Transforming WordNet-like Databases

The final publication is available at `link.springer.com`
`http://link.springer.com/chapter/10.1007/978-3-319-08958-4_28`
DOI: 10.1007/978-3-319-08958-4_28

Marek Kubis

Adam Mickiewicz University, Faculty of Mathematics and Computer Science,
Department of Computer Linguistics and Artificial Intelligence, ul. Umultowska 87,
61-614 Poznań, Poland
`mkubis@amu.edu.pl`

Abstract. The paper presents WUpdate – a data manipulation language designed for WordNet-like lexical databases. The language can be used to perform modifications of a wordnet, such as adding word senses, removing words, interlinking synsets, etc. The focus of the paper is on solving wordnet-specific problems that are not addressed by other data manipulation languages. In particular, the paper addresses the problem of preserving the properties of semantic hierarchies while they are being transformed and the problem of changing the granularity of a WordNet-like database. The paper outlines the syntax and semantics of the WUpdate language and describes the underlying data model. Alternative approaches that may be undertaken to modify a WordNet-like database are discussed.

Keywords: wordnet, data manipulation language, query language

1 Introduction

WordNet [4] and other lexical databases that store data in a structure built upon interlinked sets of synonymous words[1] (synsets) are exploited in a vast number of projects that involve natural language processing. As the majority of wordnets are general purpose databases that represent concepts from various domains of knowledge and store them at different levels of granularity, a problem arises of adjusting the structure and content of a wordnet to a particular task. Such an adjustment may involve the removal and creation of synsets and word senses, changing semantic and lexical relations, establishing a new definition of synonymy, augmenting synsets with additional data, etc. The paper presents WUpdate – a tool designed to perform such transformations while preserving invariants imposed by the structure of a WordNet-like database. WUpdate is a data manipulation language that, with respect to wordnets, plays a similar role as SQL [1] with respect to relational databases, or Lorel [2] to semistructural ones. WUpdate is based on WQuery [9] – a query language operating

[1] We call such lexical databases wordnets or WordNet-like databases in the paper.

on wordnet-specific data types that has been previously used as a module of an NLP/AI system supporting information management of mass events [8, 20]. WUpdate extends WQuery syntax with constructs responsible for modifying a WordNet-like database and ensures that the proper structure of a wordnet is preserved while transformations are applied at the semantic level. Emphasis is placed on solving wordnet-specific problems that are not considered in other data manipulation languages. In particular, the WUpdate language provides:

1. Proper handling of semantic hierarchies built with such relations as *hypernymy* and *meronymy*.
2. Special operators for managing granularity of a WordNet-like database.

The WUpdate interpreter is available for download as a part of the WQuery system distribution.[2] WUpdate inherits from WQuery the ability to import wordnets stored in the Global Wordnet Grid [5] and Wordnet-LMF [17] documents.

2 Related Work

The problem of transforming and enriching WordNet-like databases is addressed in many papers (e.g. [11, 12, 15]), but most of them do not describe the tools used for the task. Princeton WordNet website [13] enlists a wide variety of application programming interfaces designed for general purpose programming languages that could be used for the task, but these are mostly low level interfaces that provide only basic operations such as finding a synset by a sense or accessing related synsets. Hence, elaborate updates such as those described in Section 8 require considerable programming effort. In fact, most of the APIs do not provide any commands that would be responsible for modifying a wordnet. Thus, in such cases a transformation of the wordnet must involve the generation of an entirely new database.

Another possible approach to the problem is to use a data manipulation language designed for other data models such as SQL [1] or Lorel [2]. However, these languages do not incorporate wordnet-specific data types and do not provide operators responsible for adjusting the granularity of a wordnet that is analogous to those described in Section 7. Furthermore, preservation of the $Transitive$ property as defined in Section 6 requires additional programming work to be performed in these tools. The same difficulties arise if one tries to use an XML [17] or RDF [6] representation of a WordNet-like database and generate a new wordnet directly using query languages such as XQuery [3] or SPARQL [14].

Finally, wordnet editors such as DEBVisDic [7] may be used to modify a database. These editors are easier to use than our tool for local modifications, such as editing a gloss or adding a semantic link between two synsets, but they do not provide any mechanisms that would allow to apply modifications to multiple objects at once, as can be done by the WUpdate expressions shown in Section 8.

[2] See `http://www.wquery.org` for details.

If a wordnet editor is built on the client-server architecture (e.g. DEBVisDic), such serial modifications can be done via the server API. However, if the API does not incorporate a versatile query language then the custom scripts in a general-purpose programming language have to be built on the client side in order to perform complex modifications which are directly representable in WUpdate.

3 Data Model

WUpdate adopts the WQuery data model as presented in Figure 1. A WordNet-like lexical database is represented in this model by a set of domain-specific and general purpose data types (called, jointly, *basic data types*) interlinked via relations. The values of the domain-specific data types are stored in unary relations called domain sets. In Figure 1 the names of the domain sets are enclosed in parentheses and placed below the names of their corresponding data types. The basic structure of a wordnet is determined by the binary relations *synset*, *word*, *sensenum* and *pos*, which are represented in Figure 1 by edges with boldfaced labels. These relations connect a word sense to its synset, word, sense number and part of speech (POS) symbol, respectively. Additional relations may be introduced to represent data that vary among wordnets, such as the attributes of synsets (e.g. *gloss* and *nl* in Figure 1) or particular semantic and lexical relations (e.g. *meronym* and *antonym*). Although all of the relations presented in Figure 1 are either unary or binary, it should be noted that the model permits relations of arbitrary arity. Every argument of a relation in the data model has an assigned position, name and data types that it can take. For instance, the first argument of the relation *pos* as shown in Figure 1 has the name *src* and it has to take synsets (i.e. values of the *Synset* data type), and the second one is named *dst* and has to take part of speech symbols.

The data stored in an instance of the WQuery data model are described by a distinguished set of relations called the *metamodel*. The names of the relations stored in an instance of the model are gathered in the relation *relations*. The assignments of data types, positions and names to the arguments of relations are kept in the relation *arguments*. Both relations are directly accessible for the query language.

4 WQuery Language

We use the WQuery language in our tool to identify objects that have to be transformed and to define the results of a transformation. The WQuery construct we use most extensively is a path expression. It is an expression that describes a multiset of paths in an instance of the data model presented in Section 3. For the purpose of path expressions, the instance is treated as a directed graph. This graph's set of nodes consists of values of the basic data types that belong to the active domain[3] of the instance. The set of edges is given by pairs of values that

[3] I.e. the set of values referenced by at least one relation of the instance.

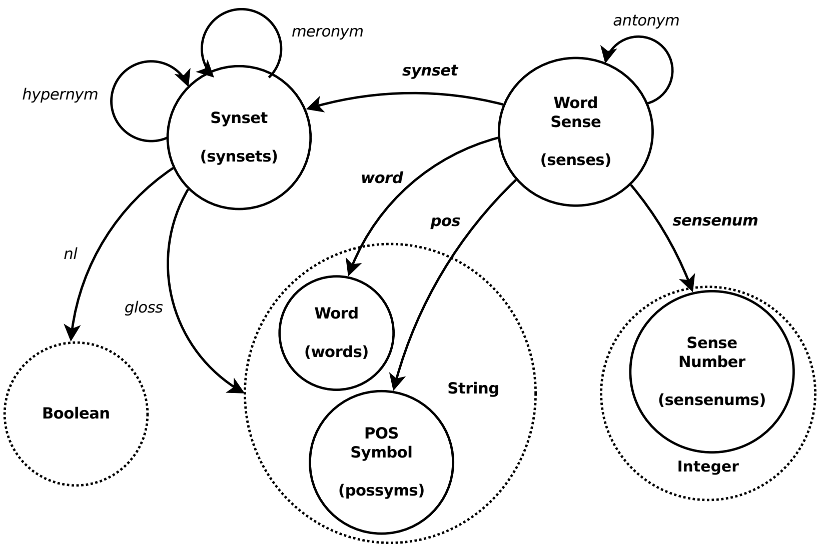

Fig. 1. Basic data types (as nodes of the graph) and the relations among them (as edges) in an instance of the WQuery data model.

belong to tuples of relations gathered in the instance. A path expression begins with a *generator*, i.e. an expression that identifies a subset of values of one of the data types represented by the nodes of the graph. The generator is followed by zero or more regular expressions formulated over the names of relations stored in the instance of the WQuery data model. A regular expression specifies which edges have to be traversed in order to extend the paths retrieved by the preceding expression. For example, the query[4]

```
{auto:1:n}.hypernym+
```

consists of a generator `{auto:1:n}` which retrieves the synset that contains the first noun sense of the word *auto* (Eng. *a car*) followed by a regular expression `hypernym+` which traverses one or more times through the edges that link hyponyms (the values of the first argument of the `hypernym` relation) to their hypernyms (the values of the second argument). Thus, the result of the query consists of paths that link `{auto:1:n}` with its transitive hypernyms through

[4] Unless stated otherwise, the queries in the paper are invoked against PolNet [19, 18] – a WordNet-like database developed for the Polish language.

zero or more intermediate nodes. Figure 2 presents some paths retrieved by the query as formulated above.

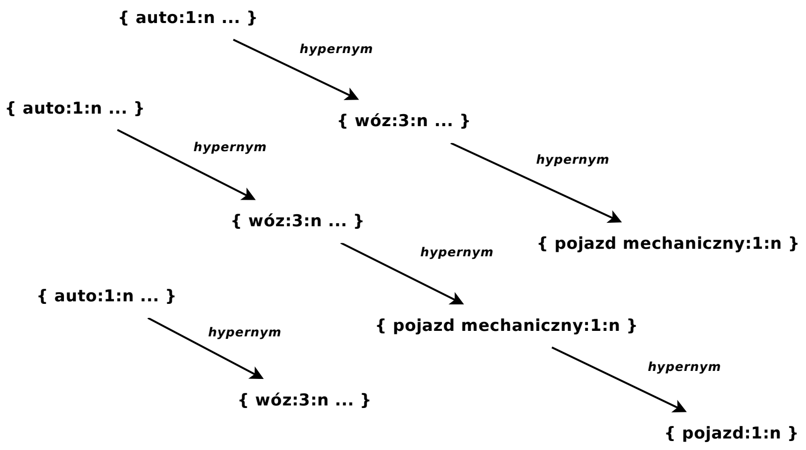

Fig. 2. Sample paths returned by the query `{auto:1:n}.hypernym+`.

In order to retrieve paths that link synsets to their transitive hyponyms by using the `hypernym` relation, one has to traverse the edges that link the values of the second argument of this relation to the values of the first argument.[5] The backward edges are accessed by prepending the `^` sign to the relation name. Hence, the paths that link `{auto:1:n}` with its transitive hyponyms are given by the following query

```
{auto:1:n}.^hypernym+
```

One can place a variable after a specific step[6] of a path expression in order to reference the values reached by the step. For example, the expression

```
{auto:1:n}$a.hypernym+$b
```

attaches bindings to every retrieved path that associate the variable `$a` with the first node on the path and the variable `$b` with the last node.

Variable bindings may be referenced in filters, i.e. bracketed conditional expressions that may be placed after the step of a path expression to eliminate undesirable paths from the query result. For instance, the following query returns paths that begin with a synset that contains the word *auto* and ends with its meronym that contains at least two word senses.

[5] In the paper we skip the methods of accessing the arbitrary arguments of non-binary relations. The details can be found in [10].

[6] By a step we mean the generator or any of the regular expressions of a path expression that follow.

```
{auto}.meronym$a[count($a.^synset) >= 2]
```

One may use the `from` expression in order to iterate through all the bindings of a path expression. For example, one may formulate the expression

```
from {auto:1:n}.hypernym+$a
  emit $a
```

to pass to the output (i.e. emit) only values that are bound to the variable `$a`.

5 WUpdate Language

A WUpdate expression consists of two WQuery expressions interleaved by a relation name and a transformation operator. The tuples of the specified relation which are determined by the left-hand expression are updated using the values determined by the right-hand expression with regard to the transformation operator.[7] For instance, the expression

```
{autobus:1:n} hypernym += {auto:1:n}
```

connects the synset *{ autobus:1:n }* (Eng. *a bus*) to the synset *{ auto:1:n }* via the relation `hypernym`. Besides the `+=` operator, which adds tuples to a relation, one can also use `-=` to remove them and `:=` to replace the tuples conforming to the objects determined by the left-hand expression with new tuples that end in the right-hand objects.

Values of wordnet-specific data types, shown in Figure 1 as nodes, may be added or removed from the database by modifying relations that correspond to their data type names. For example, to add the second noun sense of the word *auto* one may execute the following expression

```
senses += auto:2:n
```

In order to change the set of relations that are stored in an instance of the WQuery data model one has to modify the relations of the metamodel. For example, to create the relation `usage` that connects synsets to the sentences that are examples of their usage, one may execute the following commands

```
relations += ‘usage‘
‘usage‘ arguments += ‘src‘,‘synset‘,1
‘usage‘ arguments += ‘dst‘,‘string‘,2
```

The first expression adds the relation named `usage` to the instance. The second one specifies that the first argument of `usage` is named `src` and takes synsets as its values. The last one states that the second argument of `usage` is named `dst` and is restricted to string values. The comma (`,`) operator in the expressions above represents the concatenation of the paths. By default the elements of paths returned by the arguments of an update operator have to correspond to the consecutive arguments of the modified relation. Thus, in the expression

[7] The left-hand expression is optional. If it is not specified, all the tuples of the relation are considered as shown in the examples later in this section.

```
'usage' arguments += 'dst','string',2
```

the string `usage` is assigned to the first argument of the relation `arguments`, the string `dst` to the second one, etc. One can reference the arguments of a relation by name with `^` signs placed in between in order to assign values to them in an arbitrary order. Hence, the tuple added by the expression above may also be created by the command

```
'usage' relation^arguments^position^name^type += 2,'dst','string'
```

6 Preserving Relation Properties

Modifying the content of a wordnet may corrupt its structure. For example, if we add a new connection through the `hypernym` relation, we may accidentally create a cycle in the hypernymy hierarchy which by definition is acyclic. In order,to prevent such situations we have extended the WQuery data model by introducing properties that may be associated with arguments of relations in order to preserve the valid structure of a wordnet while its content is being modified. We present the available properties in Table 1. The $Required(\alpha)$, $Functional(\alpha)$, $Symmetric(\alpha, \beta)$ and $Antisymmetric(\alpha, \beta)$ properties are simple constraints that have natural counterparts in other database systems. The $Transitive(\alpha, \beta)$ property has been introduced to ease maintenance of semantic hierarchies built using such relations as hypernymy or meronymy. It may become compromised if a tuple is removed from R or if for some tuple $t \in R$ the value $t(\alpha)$ or $t(\beta)$ is removed from a domain set. In the first case the default action is to permit such an operation assuming that the change has been introduced intentionally. In the second case the tuples that contain the removed value on the β position have to be joined with the tuples that contain it on the α position, as is shown in the example in Figure 3. Hence, the removal of intermediate objects from a transitive relation preserves the reachability of values as defined in Table 1. It should be noted that in the case of non-binary relations the join operation will succeed only if the tuples being joined are equal except for the values of α and β, otherwise an error will be reported.

By default WUpdate reports an error if $Required(\alpha)$, $Functional(\alpha)$ or $Antisymmetric(\alpha, \beta)$ becomes compromised by a command submitted to the interpreter. The default action undertaken in the case of the $Symmetric(\alpha, \beta)$ property is to restore the symmetry. Thus, if a new tuple is added to a symmetric relation then the backward tuple is also created, and if a tuple is removed the backward one is removed also. If the default action associated with a property is not adequate, it may be replaced by an alternative from Table 2.

Descriptions of properties are stored in two new relations added to the metamodel:

1. *properties(relation, argument, property, action)*
   that collects properties (arg. *property*) assigned to the arguments (arg. *argument*) of relations (arg. *relation*) and the actions bound to them (arg. *action*)

Table 1. Properties of relation arguments. α, β – arguments of the relation R, $dt(x)$ data type of the argument x

Name	**Definition**	**Example**
$Required(\alpha)$	If v is a value of of the type $dt(\alpha)$ then there exists a tuple $t \in R$ such that $t(\alpha) = v$	`pos(src)`
$Functional(\alpha)$	If v is a value of of the type $dt(\alpha)$ then there exists at most one tuple $t \in R$ such that $t(\alpha) = v$	`gloss(src)`
$Symmetric(\alpha, \beta)$	if there exists a tuple $t \in R$ such that $t(\alpha) = v$ and $t(\beta) = w$ then there exists a tuple $s \in R$ such that $s(\alpha) = w$ and $s(\beta) = v$	`antonym(src,dst)`
$Antisymmetric(\alpha, \beta)$	if there exists a tuple $t \in R$ such that $t(\alpha) = v$ and $t(\beta) = w$ then there is no tuple $s \in R$ such that $s(\alpha) = w$ and $s(\beta) = v$	`meronym(src,dst)`
$Transitive(\alpha, \beta)$	if w is reachable from v through R (i.e. there exist tuples $t_1, t_2, \ldots, t_k \in R$ such that $t_1(\alpha) = v$, $t_k(\beta) = w$, $t_i(\beta) = t_{i+1}(\alpha)$ for $i < k$) before transformation and the transformation does not remove either w or v then w is reachable from v after the transformation is performed	`hypernym(src,dst)`

2. *pair_properties(relation, source, destination, property, action)*
   that collects properties (arg. *property*) of pairs of arguments (arg. *source* and *destination*) of relations (arg. *relation*) and the actions bound to them (arg. *action*)

These relations have to be modified in order to add or remove the properties of relation arguments. For instance, to make the `hypernym` relation antisymmetric, one can invoke the following command

```
‘hypernym‘, ‘src‘, ‘dst‘
    pair_properties := ‘antisymmetric‘, ‘preserve‘
```

7 Managing Granularity of a Wordnet

Since the problem of adjusting the granularity of a wordnet to a target application has been discussed in several papers (e.g. [11, 12, 15], in the context of word sense disambiguation), we decided to introduce the `split` and `merge` operators that simplify adjustment of the partition of senses into synsets.

The `split` operator takes as an argument a set of synsets and creates separate synsets for all of their senses. For example, to split the synset *{ auto:1:n samochód osobowy:1:n bryka:2:n samochodzik:2:n }* into the synsets *{ auto:1:n }*, *{ samochód osobowy:1:n }*, *{ bryka:2:n }*, *{ samochodzik:2:n }* one may formulate the following expression

Table 2. Alternative actions that may be undertaken if a property is compromised

Property	**Cause**	**Alternative Actions**
Required(α)	A tuple t has been removed	Remove the value $t(\alpha)$
Functional(α)	A tuple t has been added	Remove the tuple s such that $t(\alpha) = s(\alpha)$
Symmetric(α, β)	A tuple has been added or removed	Report an error
Antisymmetric(α, β)	A tuple t has been added such that for some tuple s holds $t(\alpha) = s(\beta)$ and $t(\beta) = s(\alpha)$	Remove the tuple s
Transitive(α, β)	$t(\alpha)$ or $t(\beta)$ has been removed for some tuple $t \in R$	Report an error

```
split {auto:1:n}
```

The `merge` operator takes as an argument a set of synsets and/or word senses and relocates senses to the new synset. For instance, to create a new synset from all synsets that contain the word *pojazd* (Eng. *a vehicle*), one may execute the following command.

```
merge {pojazd}
```

The main problem that arises while relocating senses is how to deal with the connections of their synsets. By default new synsets inherit all connections after the merged/split ones, and in the case of the `merge` operator edges that point from the new synset to the same object are joined together. Since the `split` and `merge` operators preserve relation properties as defined in Section 6, they may report an error if such an action is associated with a property. For instance, if one merges synsets that belong to different levels of the hierarchy determined by a transitive antisymmetric relation (e.g. hypernymy), one may encounter the error as shown in the example below.

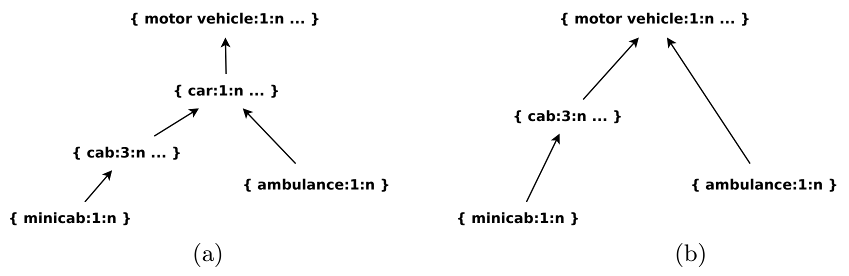

Fig. 3. The `hypernym` relation before (a) and after (b) the removal of the synset *{ car:1:n }* from the Princeton WordNet

```
merge {pojazd:1:n} union {auto:1:n}

ERROR: Update breaks property 'transitive antisymmetry'
of relation 'hypernym'
```

In order to avoid errors one has to prepend and/or append additional expressions that will guarantee that the properties of the relations will not become compromised. Due to the limited space we skip the details, which can be found in [10].

8 Examples of Complex Transformations

In order to show how the WUpdate language may be used to perform complex database transformations in this section we consider problems that require more elaborate changes to be performed on a wordnet. We have chosen problems that do not depend on the existence of particular synsets, word senses or words in the database, but which rely only on the structural properties of a wordnet, such as existence of hypernymy and meronymy relations. Thus, the WUpdate expressions presented in this section may be invoked against different WordNet-like databases without introducing adjustments.

8.1 Removing Redundant Connections

Hypernymy is a transitive relation. Hence, if two synsets are reachable via hypernymy through one or more intermediate synsets then maintaining a direct hypernymy link between them is redundant. Such links may be removed with the following expression[8]

```
from {}$a.hypernym.hypernym+$b[$b in $a.hypernym]
  $a hypernym -= $b
```

A similar problem arises if we assume that hyponyms inherit meronyms from their hypernyms. In this case we can remove the meronymy links from the hyponyms.

```
from {}$a.meronym$b[$b in $a.hypernym+.meronym]
  $a meronym -= $b
```

8.2 Separating Senses by Parts of Speech

WUpdate does not enforce synsets to be composed only from senses that belong to the same part of speech. If such a separation is necessary for a given task, it can be introduced by iterating through synsets and merging senses that belong to the same part of speech.

[8] The `{}` generator represents all synsets in the database.

```
from {}$a
  from possyms$b
    merge $a.^synset[$a.pos = $b]
```

The cross-part-of-speech hypernymy links (if any exist) can be removed with the following expression

```
from {}$a.hypernym$b[$a.pos != $b.pos]
  $a hypernym -= $b
```

9 Conclusion

The paper describes a tool designed to perform transformations of WordNet-like lexical databases. WUpdate incorporates several features that have been designed to simplify manipulation of wordnet data that are not found in other wordnet-oriented and general purpose data management tools. First, it is built upon a query language that incorporates wordnet-specific data types. Second, it adopts a data model that preserves the properties of semantic relations while the content of a wordnet is being transformed. Third, it provides a set of operators for managing the granularity of a WordNet-like database. We have shown how these features, when combined together, can be exploited to perform complex wordnet transformations.

In the future we would like to extend the language with constructs responsible for processing multiple WordNet-like databases at once. As WUpdate performance is coupled to the performance of the WQuery engine, we also plan to investigate query optimization methods that exploit the structure of a wordnet.

Acknowledgments. This research was supported by a scholarship within the project "Scholarship support for Ph.D. students specializing in majors strategic for Wielkopolska's development", Sub-measure 8.2.2 Human Capital Operational Programme, co-financed by European Union under the European Social Fund.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley (1995)
2. Abiteboul, S., Quass, D., McHugh, J., Widom, J., Wiener, J.L.: The Lorel Query Language for Semistructured Data. International Journal on Digital Libraries 1(1), 68–88 (1997)
3. Boag, S., Chamberlin, D.D., Fernández, M.F., Florescu, D., Robie, J., Siméon, J.: XQuery 1.0: An XML Query Language (Second Edition). W3C recommendation, W3C (December 2010), http://www.w3.org/TR/2010/REC-xquery-20101214/
4. Fellbaum, C. (ed.): WordNet: An Electronic Lexical Database. MIT Press, Cambridge, MA (1998)
5. Global Wordnet Association: Global Wordnet Grid DTD. `http://globalwordnet.org/gwa/grid/bwn2.dtd` (2010), access date: 23 September 2010

6. Graves, A., Gutierrez, C.: Data Representations for WordNet: A Case for RDF. In: Sojka et al. [16], pp. 165–169
7. Horak, A., Pala, K., Rambousek, A., Povolny, M.: DEBVisDic - First Version of New Client-Server Wordnet Browsing and Editing Tool. In: Sojka et al. [16], pp. 325–328
8. Kubis, M.: An Access Layer to PolNet – Polish WordNet. In: Vetulani, Z. (ed.) Human Language Technology. Challenges for Computer Science and Linguistics, Lecture Notes in Computer Science, vol. 6562, pp. 444–455. Springer Berlin / Heidelberg (2011)
9. Kubis, M.: A Query Language for WordNet-like Lexical Databases. In: Pan, J.S., Chen, S.M., Nguyen, N.T. (eds.) Intelligent Information and Database Systems, Lecture Notes in Artificial Intelligence, vol. 7198, pp. 436–445. Springer Heidelberg (2012)
10. Kubis, M.: WQuery User Guide. `http://wquery.org/user-guide.pdf` (2013)
11. Mihalcea, R., Moldovan, D.I.: EZ.WordNet: Principles for Automatic Generation of a Coarse Grained WordNet. In: Russell, I., Kolen, J.F. (eds.) Proceedings of the Fourteenth International Florida Artificial Intelligence Research Society Conference, May 21-23, 2001, Key West, Florida, USA. pp. 454–458. AAAI Press (2001)
12. Navigli, R.: Meaningful clustering of senses helps boost word sense disambiguation performance. In: Proceedings of the 21st International Conference on Computational Linguistics and the 44th annual meeting of the Association for Computational Linguistics. pp. 105–112. ACL-44, Association for Computational Linguistics, Stroudsburg, PA, USA (2006)
13. Princeton University: WordNet - Related Projects. `http://wordnet.princeton.edu/wordnet/related-projects/` (2011), access date: 28 April 2011
14. Prud'hommeaux, E., Seaborne, A.: SPARQL Query Language for RDF. W3C recommendation, W3C (January 2008), http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/
15. Snow, R., Prakash, S., Jurafsky, D., Ng, A.Y.: Learning to Merge Word Senses. In: Proceedings of the Joint Meeting of the Conference on Empirical Methods on Natural Language Processing and the Conference on Natural Language Learning. pp. 1005–1014 (Jun 2007)
16. Sojka, P., Choi, K.S., Fellbaum, C., Vossen, P. (eds.): Proceedings of the Third International WordNet Conference – GWC 2006. Masaryk University, Brno, Czech Republic (2005)
17. Soria, C., Monachini, M., Vossen, P.: Wordnet-LMF: Fleshing out a Standardized Format for Wordnet Interoperability. In: Proceeding of the 2009 international workshop on Intercultural collaboration. pp. 139–146. ACM, New York, USA (2009)
18. Vetulani, Z.: Wordnet Based Lexicon Grammar for Polish. In: Calzolari, N., et al. (eds.) Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12). European Language Resources Association (ELRA), Istanbul, Turkey (May 2012)
19. Vetulani, Z., Kubis, M., Obrebski, T.: PolNet - Polish WordNet: Data and Tools. In: Calzolari, N., et al. (eds.) Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10). pp. 3793–3797. European Language Resources Association (ELRA), Valletta, Malta (May 2010)
20. Vetulani, Z., Marciniak, J.: Natural Language Based Communication between Human Users and the Emergency Center: POLINT-112-SMS. In: Vetulani, Z. (ed.) Human Language Technology. Challenges for Computer Science and Linguistics, Lecture Notes in Computer Science, vol. 6562, pp. 303–314. Springer Berlin Heidelberg (2011)