THE PUBLISHING HOUSE
OF THE ROMANIAN ACADEMY

PROCEEDINGS OF THE ROMANIAN ACADEMY, **Series A,**
Volume 16, Number 1/2015, pp. 87–94

ROWORDNETLIB – THE FIRST API FOR THE ROMANIAN WORDNET

Stefan Daniel DUMITRESCU

Research Institute for Artificial Intelligence "Mihai Drăgănescu", Romanian Academy

The article presents RoWordNetLib, a Java Application Programming Interface (API) for the Romanian WordNet (RoWN). RoWN is a mature semantic network mirroring Princeton WordNet, and it is in continuous development by RACAI since its first version in 2004. A wordnet is a very useful resource in several domains such as Information Retrieval, Information Extraction, Word Sense Disambiguation, etc. The development of this API is meant to ease the use of the Romanian WordNet in both academic and industry environments. We implemented several distinct functionality classes: RoWN file input/output, wordnet programmatic access methods, set operations on multiple wordnet objects, word sense disambiguation basic algorithms that enable the use of distance-based and semantic similarity functions. The API is open-source, freely available and extendible.

Key words: Romanian WordNet, Java API, similarity metrics, word sense disambiguation.

1. INTRODUCTION

The Romanian WordNet (RoWN) [1, 2] is a semantic network built on the same principles and aligned with Princeton WordNet (PWN) (Fellbaum, 1998). The first version of RoWN was one of the outcomes of the BalkaNet European project[1] (2001–2004), a joint development of the RACAI[2] and UAIC[3] research teams. Since then, it has been under continuous development within the NLP group of RACAI [3].

A semantic network consists of nouns, verbs, adjectives and adverbs interlinked by lexical-semantic relations. RoWN closely follows the content and structure of PWN, being aligned with its 3.0 version. A wordnet's basic unit is the synset. A synset is a set of synonyms, defining a specific meaning, common to the members of the synset. It has additional data, like type (a noun, verb, adverb or adjective synset), definition (a sentence or small paragraph describing the general sense of the synset) and relations (named links pointing to other synsets). During its development, RoWN synsets have been enriched with associated SUMO/MILO concepts[4], labeled with WordNet Domains 3.0 categories[5], as well as containing SentiWordNet [4] information. The synonyms in a synset are called literals. A literal is a word with an associated sense in a dictionary. For example, a synset defined as "*Parte a globului pământesc, a unui continent, a unei țări etc. aşezată spre nord*" contains literals "*miazănoapte*" with sense *1.2* and "*nord*" with sense *5.1*. The sense numbering in RoWN mirrors the Explanatory Dictionary of the Romanian Language ("Dicționarul explicativ al limbii române" – DEX) [5]. However, if in PWN synsets have their literals ordered by their occurrence frequency in a corpus (meaning the first literal is the most frequent for the synset), in RoWN there is no such ordering. The senses associated with literals are borrowed from DEX and do not imply an order for the literals within a synset.

A synset is linked to other synsets through a small range of canonic relations such as hypernymy (*plant* is a hypernym of *flower* – and *flower* is a hyponym of *plant*), meronymy (part, member or substance: a *leaf* is a meronym (part-of) of a *tree*), entailment, attribute, cause, instance_hyponym, etc. While in the current version of RoWN there are only 35 different relation types, some of them affecting no more than a few

[1] http://www.dblab.upatras.gr/balkanet/
[2] Research Institute for Artificial Intelligence, Romanian Academy, Bucharest, http://www.racai.ro
[3] Faculty of Computer Science, "Al. I. Cuza" University, Iasi, http://www.uaic.ro/uaic/bin/view/Main/WebHome
[4] www.ontologyportal.org
[5] wndomains.fbk.eu/labels.html

synsets, in total there are 196,156 relations between 59,348 synsets. Interestingly, some relations give rise to important data structures. Such a relation is hypernymy, which is a hierarchical relation and creates the "hypernym tree", a tree-like graph (it does contain a few loops that break the tree structure, but for all intents and purposes it can be viewed as a tree). This tree is practically a taxonomy of meanings. Starting with a root element, each child is a more specific meaning than its parent. The data structure is used to measure the closeness of words, to find out a common parent of two or more different concepts, to link instances to their class, etc.; the "hypernym tree" is essential in a significant number of NLP applications like Information Retrieval, Information Extraction, Word Sense Disambiguation, etc.

Although RoWN is a mature resource, currently there is no Application Programming Interface (API) for it. Such a tool is essential if RoWN is to become a real presence in Romanian-enabled smart applications. For example, for PWN there is a large number of APIs written in various languages that provide a programming interface.

For Java, there are several interfaces (http://wordnet.princeton.edu/wordnet/related-projects/#Java), each having different strengths and weaknesses: JWNL/extJWNL[6], RitaWN[7], JAWS[8], WNJN[9], JWNL[10], JWI[11], etc. Some of them have the wordnet already included and ready for use; others need the dictionary files as input. The different APIs cover different user needs: JWI offers an interesting and versatile interface, having the downside that it is rather complex and thus difficult to extend; at the other end of the spectrum, RitaWN is very simple to use, but lacks some functionalities such as the ability to perform breadth-first searches on the semantic network.

Given this plentiful possible API choices, one might ask why an already-existing API cannot be used with RoWN instead of PWN. The main issue with most of the current interfaces is that they are focused on PWN, both in the input/output functionalities (e.g., PWN is usually either embedded in the interface or is distributed as "dict" (dictionary) files, whereas RoWN is distributed as a single XML file) and in the data structures within the interface, making it difficult to adapt programmatically (RoWN synset data differs from PWN synset data: with a couple of exceptions, RoWN contains all PWN data fields, as well as extra ones). Also, it is easier to add a new functionality on top of a completely open, RoWN-designed API than to reverse engineer some of the code of existing PWN APIs.

The article focuses on the technical description of the RoWordNetLib API. We present its architecture and functionality (Section 2), a programmatic evaluation of the API (Section 3), closing with a set of conclusions (Section 4).

2. THE ROWORDNET API

RoWordNetLib is structured into several packages, each with its assigned functionality (Table 1).

Table 1

Packages of RoWordNetLib

Package	Description
data	Defines required data structures
io	Provides input/output functionality
op	Graph walk functions, set operations, similarity metrics
utils	Text preprocessing utilities, timing utilities
wsd	Word sense disambiguation functionality

Before presenting the packages in detail, to avoid naming confusion, we agree to the following notations: RoWN refers to the Romanian WordNet; RoWordNetLib is the name of the API; RoWordNet is an object (in its programming language sense, a Java object) containing the wordnet. As an example, we can

[6] http://extjwnl.sourceforge.net/
[7] http://rednoise.org/rita/wordnet/documentation/index.htm
[8] http://engr.smu.edu/%7Etspell/
[9] http://wnjn.sourceforge.net/
[10] http://sourceforge.net/projects/jwordnet
[11] http://projects.csail.mit.edu/jwi

say that one can use RoWordNetLib to instantiate several RoWordNet objects, each containing different versions of RoWN.

2.1. Package *‘data’*

The *data* package contains the data structures RoWordNetLib uses internally. Its structure is rather simple: a RoWordNet object contains an array of Synset objects which are indexed by the synset ID for retrieval speed. Each Synset object contains a number of primitive types (as shown in Table 2) as well as an array of Literal objects and an array of Relation objects.

Table 2

Synset structure (main data fields)

Data field	**Description**
id	The unique ID each synset possesses in the WordNet. Ex: “ENG30-00034213-n” denoting a PWN3.0 imported noun synset with id 34213. The ID field can take any value.
pos	The type of synset: Noun (n), Verb (v), Adverb (r) or Adjective (a)
literals	An array of Literal objects
relations	An array of Relation objects
definition	A string containing the definition of a synset. Example: the synset defined by literals “compromis” and “concesie” has the following definition: “punct asupra căruia s-a făcut o concesie”
domain	The synset’s domain. Example: the synset defined by literal “închisoare” is in the “law” domain.
sumo, sumotype	As RoWN was extended with SUMO [6] information, each synset now has SUMO information. Example: the previous “închisoare” synset has sumo = “Building” and type = “+”. There are three possible types, “+” means that the sumo concept is a “hypernym” of the synset, “=” means that the sumo concept is equivalent to the synset itself, and “@” means that the synset is an “instance of” the concept.
Sentiwn_p/n/o	Similarly, RoWN was extended with SentiWordNet [3] information. Each synset has a SentiWordNet Positive, Negative and Objective value, ranging from 0 to 1.
nl	Denotes a non-lexicalized synset (a synset which could not be translated from PWN to RoWN as the word does not exists in Romanian). Example synset: “ENG30-03937437-n” with definition: “Cutie po□tală care stă în picioare”.

A Literal object contains a word and an associated sense. A Relation object contains a relation (string) that points to a target synset (defined as an ID), as well as optionally having a source and target literal for cases where the relation is not between synsets but between two synsets’ particular literals.

Figure 1 presents a synset having a single literal and 4 different relations.

```
<SYNSET>
  <ID>ENG30-01473806-n</ID>
  <POS>n</POS>
  <SYNONYM>
        <LITERAL>vertebrat_acvatic<SENSE>1</SENSE></LITERAL>
  </SYNONYM>
  <ILR>ENG30-01471682-n<TYPE>hypernym</TYPE></ILR>
  <ILR>ENG30-01474283-n<TYPE>hyponym</TYPE></ILR>
  <ILR>ENG30-02465084-n<TYPE>part_meronym</TYPE></ILR>
  <ILR>ENG30-02512053-n<TYPE>hyponym</TYPE></ILR>
  <DEF>Animal vertebrat care trăie□te în ape</DEF>
  <DOMAIN>animals</DOMAIN>
  <SUMO>Vertebrate<TYPE>+</TYPE></SUMO>
  <SENTIWN><P>0</P><N>0</N><O>1</O></SENTIWN>
</SYNSET>
```

Fig. 1 – Synset XML format example.

2.2. Package *‘io’*

The *io* package provides input and output functions. The most important I/O function is reading and writing RoWordNet objects in their native XML format. The XMLRead and XMLWrite classes offer simple static functions.

```
RoWordNet rown = new RoWordNet(XMLRead.read("wn_rev35.xml"));
XMLWrite.write(rown, "wn_rev36.xml", true, false);
```

Fig. 2 – Reading and writing RoWordNet objects in XML format.

A RoWordnet object is created by simply initializing it with the output of the XMLRead.read() function. Writing offers two extra Boolean flags: overwrite automatically and compressed output. In figure 2 the last parameter of the write function is false, specifying that the output should be written in human-readable format, as presented in Fig. 1. Compressed output removes empty lines and tabs, minimizing space requirements.

The *io* package also provides console printing and RoWordNetLib debug methods as well as general file reading methods which can safely read UTF8 files that have the Byte-Order-Mark (BOM) set. The BOM is a problem for many parsers - they fail unless the BOM is removed or skipped over.

2.3. Package *‘op’*

The *op* package provides a number of different operational tools on RoWN. We first present the set operations, present in the Operation.java class (Table 3). All functions receive two RoWordNet objects, named o1 and o2.

Table 3

Set operations on RoWordNet objects

Method	**Description**
`intersection`	Returns a list of synset IDs that are present (and equal) both in o1 and o2.
`union`	Returns a RoWordNet object containing the union of the o1 and o2 RoWordNet objects. If there is any synset with the same ID in both o1 and o2 but with different content, the method throws an exception - the new RoWordNet cannot contain two different synsets with the same ID.
`merge`	The merge function adds any synsets in o2 over those in o1, returning o1. If there are two synsets with the same ID, it overwrites the one in o1 with the one in o2 automatically.
`complement`	Returns the complement of o2 in o1: the list of synset IDs that are in o1 but not in o2
`difference`	Returns a list of synset IDs that are present both in o1 and o2 (same ID) but have different contents

These methods are useful in several scenarios. For example, if a number of people are working in parallel but offline (not communicating between them – not working on a single RoWordNet object) and then need to synchronize their changes (deletions, additions and synset editing) into a single final RoWordNet, without such set operations the integration process would be slow and error-prone.

The *op* package provides, through the BFWalk class, the ability to iterate programmatically through the RoWN semantic network in a breadth-first fashion. BFWalk’s constructor takes as input the RoWordNet object it will iterate through (1), a starting (source) synset (2), a Boolean variable specifying whether all relations are allowed as valid edges during the iteration (3) and an array of “filtered” relations (4). The constructor functions differently depending on (3): if the Boolean is set to true, then the relations given in (4) are not accepted as valid edges during the walk (default: if no relation is set in (4) then all are accepted); if the Boolean is set to false, then the only allowed relations are those specified in (4).

Finally, the *op* package implements a number of similarity measures through the SimilarityMetrics class. There are two distinct types of similarity measures: distance-based and semantic.

The distance based measure is simply the number of edges between two synsets. The closer the synsets are, the more similar they are said to be. The measure is optimized to use two synchronous breadth-first searches starting from the source and target synsets.

The semantic measures function on the concept of Information Content (IC), a concept introduced by Resnik in 2005 [7]. Information Content is a specificity measure for concepts: concepts that are more specific have a higher IC value than more general concepts (ex: locomotive has a higher IC than device). The

IC value is calculated depending on the frequency of concepts from the text. The process is as follows: the text (corpus) from which IC values are to be derived from is parsed, and for every concept found, its frequency as well as the frequency count of its ancestors are increased by one in WordNet. This directly introduces the problem of Word Sense Disambiguation (treated in section 2.4): a word from the text should correctly identify the sense (synset) to which it will contribute – wrongly picking senses will deteriorate the IC count and directly the semantic similarity scores. The IC value is computed as the negative log of the probability (frequency count) of the concept: IC(synset) = –log(P(synset)). Package *wsd* provides the methods through which the IC is computed from any given corpus, enabling usage of the semantic similarity measures.

The SimilarityMetrics class implements three semantic similarity measures (Table 4). Before presenting them, we need to define the concept of the Least Common Subsumer (LCS): as all synsets are linked to other synsets through semantic relations, then one can find for two arbitrary synsets the closest “parent” synset – the LCS. This concept is defined in the context of WordNet being a tree, with synsets as nodes and the hypernym (sometimes also meronym) relations as edges.

Table 4

Implemented semantic similarity measures

Sim Measure	Description	API Methods
$\mathbf{Sim_{RES}}$ (Resnik - 1995)	$sim_{RES}(s_1,s_2) = IC(LCS(s_1,s_2))$	`SimilarityMetrics.`**`Resnik`**

$\mathbf{Sim_{JCN}}$ (Jiang and Conrath - 1997)	$dist_{JCN}(s_1,s_2) = 2*sim_{RES} - IC(s_1)-IC(s_2)$
	 $sim_{JCN} = 1/dist_{JCN}$
		`SimilarityMetrics.`**`JiangConrath_distance`**
		 `SimilarityMetrics.`**`JiangConrath`**

$\mathbf{Sim_{LIN}}$ (Lin - 1998)	$sim_{LIN} = (2*sim_{RES})/(IC(s_1)+IC(s_2))$	`SimilarityMetrics.`**`Lin`**

The Resnik measure (RES) provides the basic metric that is used both for LIN and JCN measures. The similarity value is the Information Content value of the synset’s LCS. The RES measure may provide the same value for different synsets that share the same LCS, and thus might not be very informative. Lin’s measure attempts to improve the accuracy by incorporating information about the IC of each of the synsets. Jiang and Conrath provide an alternate distance metric instead, using the same elements as Lin. However, to transform JCN into a similarity measure, one has to simply invert the distance formula.

These three measures types represent standard measures used for a long time in NLP applications, with consistent results. In our implementation, all the methods are static and have four parameters: the RoWordNet object (parameter #1), the two synsets (#2&#3) and a Boolean specifying if all relations should represent valid edges on which to find the LCS (parameter #4). Making the Boolean false will force the standard LCS, found by traversing only the hypernym and instance_hypernym relations.

2.4. Package *‘wsd’*

The *wsd* package is used to compute the IC value given an arbitrary Romanian text corpus and a RoWordNet object. The IC is obtained by using one of the two WSD algorithms, both implementing the generic computeIC() method offered by the WSD interface. The two unsupervised WSD algorithms are Lesk [8] and an adapted version of Lesk.

The classic Lesk algorithm attempts to identify word senses based on evaluating the overlapping among their sense definitions (the synsets' glosses). Give *n* words, each word having an arbitrary number of senses (possible matching synsets), the algorithm evaluates all combinations of senses, for each combination measuring the overlap in the chosen senses' definitions. The overlap is simply the number of words found in both definitions of the evaluated senses. The initial precision reported by Lesk in 1986 was around 50-70% on the English WordNet. For the Romanian WordNet we cannot offer a WSD precision as currently there is no sufficiently large RoWordNet annotated corpus on which to test.

The adapted Lesk algorithm is based on Banerjee and Pedersen work [7]. To disambiguate a word given an n-word sentence we first apply a k-length window to the left and to right of the target word (as this will dramatically reduce computing time given the exponential nature of the algorithm). Then, each word in the 2*k+1 window has is possible senses (target synsets) extracted. Next, for each target synset found we find all connected hypernym, hyponym, meronym and troponym synsets. For each category of connected synsets we create a single definition by concatenating all their definitions (e.g. for a given target synset having 3 hypernym connected synsets we create a "hypernym" definition composed of joining the three hypernym definitions). To evaluate the relatedness of two target synsets we simply count the overlap of all possible combinations of the definitions belonging to each of the target synsets [9] (e.g. score(s1,s2) = overlap(*gloss*(s_1), *gloss*(s_2)) + overlap(*hyper*(s_1), *hyper*(s_2)) + … + overlap(*gloss*(s1), *mero*(s_2)) + …).

The overlap function is also improved to score common words based on ZipF's Law (Wikipedia excerpt: "the law states that given some corpus of natural language utterances, the frequency of any word is inversely proportional to its rank in the frequency table. Thus the most frequent word will occur approximately twice as often as the second most frequent word, three times as often as the third most frequent word, etc."). Thus, measuring the overlap between two definitions is reduced to the problem of finding the longest common substring with maximal consecutives.

WSD performance depends heavily on the correct identification of (1) possible synsets that might represent corpus words and (2) matching lemmas of different synsets. Furthermore, running time is affected by (3) discovering possible synsets for corpus words that belong to a different grammatical category than the word itself (ex: considering verb, adverb and adjective synsets for a noun). All three issues are given a partial solution through the use of a Part of Speech (POS) Tagger and a Lemmatizer. In the WSD process the POS tagger is used to determine if a word is a noun, verb, adverb or adjective, and reject synsets that do not match its category (issue 3). Next, all words in the corpus are lemmatized. This ensures a significantly higher word recognition rate (issue 1 – for example, the definite plural noun "brazii" in a corpus will never match the literal "brad", given that "brad" is singular, non-definite noun; lemmatization will transform "brazii" into "brad", thus making the synset association possible). Finally, in the definition matching process between synsets, the definitions themselves use inflected words – the lemmatized version of the definitions (for all grammatical categories) ensure a higher match percentage (issue 2), leading to better WSD accuracy and in turn to better IC values and similarity metrics scores.

For the POS Tagging and Lemmatization processes we use the Bermuda [10] platform. It is a complex platform offering speech and text processing functionality. Packaged as a Java jar, it is attached to RoWordNetLib as an external resource. Models for POS Tagging and Lemmatization used by Bermuda are included by default in the API. Newer versions of Bermuda, including updated models can be freely downloaded[12].

2.5. Package *'utils'*

The *utils* package provides a number of helper classes and methods. The StopWord class offers a static `removeStopWords` function that, given an array of tokens, will remove any stopwords from them. A stopword is a word with little to no informational value, such as: "ăla", "şi", "noştri", "vreo", etc. The current list contains 1281 stopwords. Another text preprocessing helper class is Tokenizer, offering string tokenization options.

[12] http://sourceforge.net/projects/bermudatexttospeech

3. EVALUATION

RoWordNetLib was designed to be as simple as possible, a design choice that also facilitates future development of the API. As such, most data types are native. Memory wise, a RoWordNet object will use (with the current version of the wordnet which is a ~32MB XML file) around 92 MB of RAM. Internally, the RoWordNet object uses a hash map of synsets (~86MB), an ordered array list of the same synsets (~83MB) and a hash map of literals (~6MB). The synset map is used to obtain O(1) access to a synset given a synset ID, the array list is kept to maintain the synset's order when reading and writing (and given that the synsets are shared between the map and the array, only a very small memory overhead is introduced), and the literal map is used to index the literals (just the words without the senses) to a list of possible synsets. The literal map is used to speed up the WSD process.

Overall, the memory requirements of RoWordNetLib are very small; it can run easily even on Java-enabled mobile devices, given that almost all devices today have at least 1 GB of RAM.

Moving on to the operational time of the API, Table 5 presents a number of operations and the time required to complete them.

Table 5

A few operational time examples

Operation	Time
Load a RoWordNet object into memory from an XML file	2s
Iterating through all the synset IDs contained in the hash map and then requesting the synset by its ID by two different methods (1: RoWN method rown.`getSynsetById`(synsetID) and 2: directly from the hash map rown.synsetsMap.`get`(synsetID))	~150ms
Searching for and displaying an exact literal (rown.containsLiteral(literal))	~20ms
Extracting all noun synsets into a new hash map (rown.`getHashedSynsetsByPos`(Synset.Type.*Noun*);)	~10ms
Fetching glosses for a word (a literal that has no sense so it could match any literals)	~10ms
Extracting all verb synsets into a new array list (rown.`getListedSynsetsByPos`(Synset.Type.*Verb*);)	1s
Generating a new (unique) ID for the first time, including prefix and suffix data (rown.`getNewId`("GetNew-", "-demo");)	~150ms
Generating sequential IDs (using the same method the second time caches the data and runs significantly faster)	~2ms
Adding a new synset to a RoWN object	~5ms
Writing a RoWN object to a file (XML format)	800ms

Any timing below 150 ms was noted with a "~" approximate notation, given that the running times are so short that sometimes the same operation takes 5ms and then it takes 20 ms. The "~" stands for the average value obtained programmatically over 100 runs. Between runs the memory was flushed as best as possible so no caching can occur. If the JVM is allowed to cache the data, a significant number of operations are performed at least twice as fast (e.g. reading takes ~1s instead of 2, extracting the verbs into an array takes ~400 ms, etc.).

Another distinct functionality of the API is the set operations: reunion, intersection, complement, difference and merging. These binary operator methods facilitate the continual work on RoWN. A key problem in its development is that when multiple persons work to improve it, they eventually come to the point of merging their distinct modifications into a single master RoWN. So far, all this synchronization and versioning work has been manually performed, a task that took an unnecessary amount of time. Furthermore, this manual synchronization does not scale to more than a few people. Our final goal from this point of view is implementing a separate application that will perform automated versioning using the set operations provided by the API.

4. CONCLUSIONS

This technically oriented article presented the first purposely-built Application Programming Interface for the Romanian WordNet, named RoWordNetLib. The API's design is based on the following principles: 1. it should be easy to use; 2. it should be easy to extend, and 3. it should offer a sufficiently large array of functionalities (while at the same time striking a balance with principles 1 and 2). The programming language of choice is Java, at present being the most widely-used language in academic research; for the industry sector, where Java still disputes the top place with C++ and C# for local applications and PHP, Ruby and Python for cloud applications, it is still a very good choice, as Java can easily be translated in C# or packaged into an online service for online applications.

The main functionalities RoWordNetLib provides are:

– Input/Output for working with XML-based RoWN files;

– Methods for working with the semantic network itself (RoWordNet objects containing RoWN);

– Set operations for working with multiple RoWordNet objects (reunion, intersection, complement, difference, merge, etc.);

– Basic Word Sense Disambiguation (WSD) algorithms;

– Similarity Metrics (both distance-based and semantic).

RoWordNetLib is meant as a tool to aid quick implementations of RoWN into both research-oriented and industry applications. The API's uses can be classified as (1) internal – help facilitate the continuous work on enriching RoWN and (2) external – Quicken the development of Romanian-enabled smart applications. By providing set operations like difference, intersection or reunion on RoWordNet objects, multiple people can work in parallel on RoWN and then easily join their different RoWN versions into a master wordnet, thus easing its development. Externally, wordnets are being successfully used to perform word sense disambiguation [11], information retrieval, information extraction, machine translation, automatic text classification and summarization. For example, the emergent properties of the semantic network make a wordnet a powerful resource for search engine query expansion or for word similarity measures.

The RoWordNetLib API is free and can be obtained from RACAI's website (www.racai.ro) or from SourceForge (https://sourceforge.net/projects/rowordnetlib/). The Romanian WordNet (RoWN) is licensed through META-SHARE (http://ws.racai.ro:9191/repository/browse/18/). It is free for academic research but restricted for commercial use.

REFERENCES

1. TUFIŞ, D., *Ro-WordNet: ontologie lexicală pentru limba română*, Academica, XVIII (208–209): *30–34*, februarie-martie 2008. ISSN 1220–5737.
2. TUFIŞ, D., BARBU E., BARBU MITITELU V., ION R., BOZIANU L., *The Romanian Wordnet*, Romanian Journal of Information Science and Technology, **7**, *1–2*, 2004, pp. 107–124.
3. BARBU MITITELU, V., *Reţea semantico-derivaţională pentru limba română*, Edit. Muzeul Literaturii Române, Bucureşti, 2013.
4. BACCIANELLA, S., ANDREA E., FABRIZIO S. *SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining.,* LREC, **10**, 2010.
5. TUFIŞ, D., BARBU MITITELU, V., ŞTEFĂNESCU, D., ION, R., *The Romanian Wordnet in a Nutshell*, Language Resources and Evaluation, **X**, *10*, 2013.
6. NILES, I., PEASE, A., *Towards a standard upper ontology*, Proceedings of the International Conference on Formal Ontology in Information Systems, 2001.
7. RESNIK, P., *Using information content to evaluate semantic similarity in a taxonomy*, 14th International Joint Conference on Artificial Intelligence, Montreal, 1995.
8. LESK, M. (1986), *Automatic sense disambiguation using machine readable dictionaries: How to tell a pine cone from an ice cream cone*, 5th SIGDOC, New York, pp. 24–26.
9. BANERJEE S., PEDERSEN T. *Extended gloss overlaps as a measure of semantic relatedness*, 2003.
10. BOROŞ, T., ŞTEFĂNESCU, D., ION, R. *Handling Two Difficult Challenges for Text-to-Speech Synthesis Systems: Out-of-Vocabulary Words and Prosody: A Case Study in Romanian*, Where Humans Meet Machines, Springer New York, 2013, pp. 137–161.
11. ION, R., TUFIŞ, D., *Multilingual Word Sense Disambiguation Using Aligned Wordnets*, Romanian Journal on Information Science and Technology, Special Issue on BalkaNet, **7**, pp. 183–200, 2004.

Received December 11, 2013