WWDS APIs: Application Programming Interfaces for Efficient Manipulation of World WordNet Database Structure

Hanumant Redkar[1], Sudha Bhingardive[1], Kevin Patel[1], Pushpak Bhattacharyya[1]
Neha Prabhugaonkar[2], Apurva Nagvenkar[2], Ramdas Karmali[2]
[1]Indian Institute of Technology Bombay, Mumbai, India
[2]Goa University, Goa, India
{hanumantredkar, bhingardivesudha, kevin.svnit, pushpakbh}@gmail.com
{nehapgaonkar.1920, apurv.nagvenkar, ramdas.karmali}@gmail.com

Abstract

WordNets are useful resources for natural language processing. Various WordNets for different languages have been developed by different groups. Recently, World WordNet Database Structure (WWDS) was proposed by Redkar et. al (2015) as a common platform to store these different WordNets. However, it is underutilized due to lack of programming interface. In this paper, we present *WWDS APIs*, which are designed to address this shortcoming. These WWDS APIs, in conjunction with WWDS, act as a wrapper that enables developers to utilize WordNets without worrying about the underlying storage structure. The APIs are developed in PHP, Java, and Python, as they are the preferred programming languages of most developers and researchers working in language technologies. These APIs can help in various applications like machine translation, word sense disambiguation, multilingual information retrieval, *etc*.

Introduction

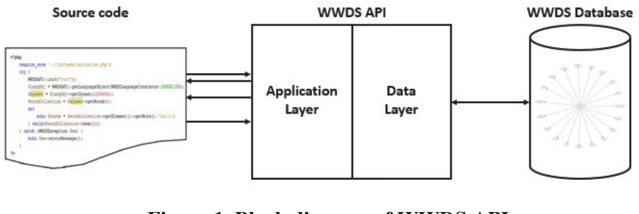
WordNet is a lexical resource primarily used in many natural language processing applications. Over a period of time, WordNets for many languages have been developed. Some of these are individual language WordNets *viz*., Princeton WordNet (Miller, 1990), Hindi WordNet, GermaNet, Japanese WordNet, *etc*. and multilingual WordNets *viz*., EuroWordNet (Vossen et al., 1997), IndoWordNet (Bhattacharyya, 2010), *etc*. Recently, a World WordNet Database Structure (Redkar et. al, 2015) has been introduced to store WordNet data in a systematic and efficient manner. However, this is not being used to its full potential due to unavailability of application programming interfaces. Hence, we present the WWDS APIs[1] to efficiently manipulate this WWDS data. These APIs will facilitate proper utilization of WWDS. For example, developers can potentially extract information from other WordNets through WWDS and its APIs that is missing in their source WordNet. The WWDS and WWDS APIs are explained in the following sections.

World WordNet Database Structure

WWDS is an efficient storage mechanism which uses multiple databases to accommodate different WordNets. Its design is based on IndoWordNet database structure (Prabhu et al., 2012). The language independent information such as semantic relations, ontology details, *etc*. is stored in a single master database named *wordnet_master*. The language dependent information such as synsets, words, lexical relations, *etc*. is stored in language specific databases named *wordnet_<language>*.

WWDS Application Programming Interfaces

WWDS APIs are developed as an extension to IndoWordNet APIs (Prabhugaonkar et al., 2012). The main objective of these APIs is to facilitate storage, retrieval and manipulation of WordNet data of all languages available in WWDS. These are developed for PHP, Java as well as Python. Each API has two layered architecture *viz*., Application Layer and Data Layer, as shown in figure 1.



```
<?php
    require_once "../lib/wwds/initialize.php";
    try {
        WWDSAPI::init("root");
        $langObj = WWDSAPI::getLanguageObject(WWDSLanguageConstants::ENGLISH);
        $synset = $langObj->getSynset([illegible]);
        $wordCollection = $synset->getWords();
        do{
            echo $words = $wordCollection->getElement()->getWord()."<br/>";
        } while($wordCollection->next());
    } catch (WWDSException $ex) {
        echo $ex->errorMessage();
    }
?>
```

Figure 1. Block diagram of WWDS API

[1] http://www.cfilt.iitb.ac.in/wwds/

Developers can call only the methods of application layer in their source code. These methods are equipped to access and manipulate WWDS information such as synsets, words, semantic and lexical relations, *etc*. However, the application layer cannot access the stored data directly, and has to rely on the data layer. The data layer provides underlying storage aware mechanisms that can directly access and manipulate the stored data. In object oriented terminology, the application layer *abstracts* the data layer. This segregation enables administrators to change the data layer according to changes in storage mechanisms, while keeping the application layer uniform.

Table 1 lists some of the major classes of WWDS APIs. For more details of these classes and the corresponding methods, please refer to the documentation[2]. Figure 2 shows sample usage snippets in all three languages.

Application Layer	
Classes	**Description**
WWDSAPI	initializes the WWDS API library
WWDSLanguage	selects the language WordNets
WWDSSynset	represents a single synset
WWDSSynsetCollection	represents a collection of synsets
WWDSWord	represents a word
WWDSWordCollection	represents a collection of words in a synset
WWDSExampleCollection	represents a collection of examples in a synset
WWDSOntology	represents an ontology node
WWDSOntologyCollection	represents a collection of ontology nodes
WWDSException	encapsulates exceptions
Data Layer	
WWDSDb	represents a database
WWDSCon	represents a connection to a database
WWDSStatement	represents data manipulation statements required by the application layer
WWDSResult	represents returned results

Table 1. Major classes of WWDS API

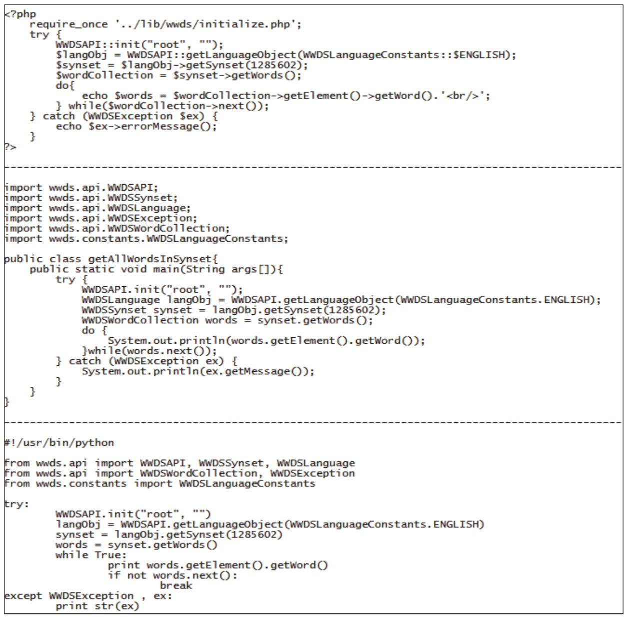
Advantages

- A single interface to access multiple WordNets, each of which could potentially be in different formats.
- Availability in popular programming languages *viz*., PHP, Java and Python ensures greater coverage of developer and researcher base.
- Data layer is adaptable to different storage mechanisms.

Limitations

- Actual usage depends on the availability (and licensing) of WordNets that developers want to use.
- Lack of authentication techniques for data modifications.

[2] http://www.cfilt.iitb.ac.in/wwds/wwdsapi/documentation/



```
<?php
    require_once '../lib/wwds/initialize.php';
    try {
        WWDSAPI::init("root", "");
        $langObj = WWDSAPI::getLanguageObject(WWDSLanguageConstants::$ENGLISH);
        $synset = $langObj->getSynset(1285602);
        $wordCollection = $synset->getWords();
        do{
            echo $words = $wordCollection->getElement()->getWord().'<br/>';
        } while($wordCollection->next());
    } catch (WWDSException $ex) {
        echo $ex->errorMessage();
    }
?>
------------------------------------------------------------------------------------------------
import wwds.api.WWDSAPI;
import wwds.api.WWDSSynset;
import wwds.api.WWDSLanguage;
import wwds.api.WWDSException;
import wwds.api.WWDSWordCollection;
import wwds.constants.WWDSLanguageConstants;

public class getAllWordsInSynset{
    public static void main(String args[]){
        try {
            WWDSAPI.init("root", "");
            WWDSLanguage langObj = WWDSAPI.getLanguageObject(WWDSLanguageConstants.ENGLISH);
            WWDSSynset synset = langObj.getSynset(1285602);
            WWDSWordCollection words = synset.getWords();
            do {
                System.out.println(words.getElement().getWord());
            }while(words.next());
        } catch (WWDSException ex) {
            System.out.println(ex.getMessage());
        }
    }
}
------------------------------------------------------------------------------------------------
#!/usr/bin/python

from wwds.api import WWDSAPI, WWDSSynset, WWDSLanguage
from wwds.api import WWDSWordCollection, WWDSException
from wwds.constants import WWDSLanguageConstants

try:
        WWDSAPI.init("root", "")
        langObj = WWDSAPI.getLanguageObject(WWDSLanguageConstants.ENGLISH)
        synset = langObj.getSynset(1285602)
        words = synset.getWords()
        while True:
                print words.getElement().getWord()
                if not words.next():
                        break
except WWDSException , ex:
        print str(ex)
```

Figure 2. WWDS API usage snippets in PHP, Java and Python

Conclusion and Future Work

Multiple WordNets use various data organization and storage methods. WWDS was developed to provide a common platform to work with multiple WordNets. However, lack of programming interface prevented its proper utilization. WWDS APIs were developed to address this shortcoming. Modular design and availability across preferred languages such as PHP, Java and Python, are some of the salient features of these APIs. However, their support of WordNet manipulation is unchecked. In the future, we would like to implement a crowd-sourcing module that can score manipulations done by the APIs, thereby resolving this limitation.

References

Bhattacharyya, P. 2010. IndoWordNet. *Proc. of LREC-10*, Malta.

Miller, George A., R., Fellbaum, C., Gross, D., & Miller, K. J. 1990. Introduction to wordnet: An on-line lexical database. *International journal of lexicography*, OUP. (pp. 3.4: 235-244).

Prabhu, V., Desai, S., Redkar, H., Prabhugaonkar, N., Nagvenkar, A., & Karmali, R. 2012. An Efficient Database Design for IndoWordNet Development Using Hybrid Approach. *COLING 2012*, Mumbai, India. (pp. 229).

Prabhugaonkar, N., Nagvenkar, A., & Karmali, Ramdas N. 2012. IndoWordNet Application Programming Interfaces. *COLING 2012*, Mumbai, India. (pp. 237 - 244).

Vossen, P. 1997. EuroWordNet: A multilingual database for information retrieval. *DELOS*, Zurich. (pp. 5-7).

Redkar, H., Bhingardive, S., Kanojia, D., & Bhattacharyya, P. 2015. World WordNet Database Structure: An Efficient Schema for Storing Information of WordNets of the World. *AAAI 2015*, Austin, Texas, USA. (pp. 4290-4291).