# IndoWordNet Application Programming Interfaces

*Neha R Prabhugaonkar*[1] *Apurva S Nagvenkar*[1] *Ramdas N Karmali*[1]
(1) GOA UNIVERSITY, Taleigao - Goa
nehapgaonkar.1920@gmail.com, apurv.nagvenkar@gmail.com, rnk@unigoa.ac.in

ABSTRACT
Work is currently under way to develop WordNet for various Indian languages. The IndoWordNet Consortium consists of member institutions developing their own language WordNet using the expansion approach. Many tools and utilities have been developed by various institutes to help in this process. In this paper, we discuss an object oriented Application Programming Interface (API) that we have implemented to facilitate easy and rapid development of tools and other software resources that require WordNet access and manipulation functionality. The main objective of IndoWordNet Application Programming Interface (IWAPI) is to provide access to the WordNet resource independent of the underlying storage technology. The current implementation manipulates data stored in a relational database. Furthermore the IWAPI also supports parallel access and manipulation of WordNets in multiple languages.

In this paper, we discuss functional requirements, design and the implementation of IndoWordNet API and its uses.

KEYWORDS: WordNet, Application Programming Interface (API), WordNet CMS, IndoWordNet, IndoWordNet Database, WordNet Website.

*Proceedings of the 3rd Workshop on South and Southeast Asian Natural Language Processing (SANLP)*, pages 237–244,
COLING 2012, Mumbai, December 2012.

## 1 Introduction

An Application Programming Interfaces (API) is defined as a set of commands, functions and protocols which developer can use when building software. It allows the developer to use predefined functions to interact with systems, instead of writing them from scratch. APIs are specially crafted to expose only chosen functionality and/or data while safeguarding other parts of the application which provides the interface. The characteristics of good API (Joshua Bloch, 2007) are as follows:

- Easy to learn and use, hard to misuse.
- Easy to read and maintain code that uses it.
- It is programming language neutral.
- Sufficiently powerful to support all computational requirements.

The IndoWordNet API provides a simple and easy way to access and manipulate the WordNet resource independent of the underlying storage technology. The functionality is exposed through a set of well defined objects that developer can create and manipulate as per his/her processing requirement. Although the current implementation expects the data to be available in a relational database, a two layered architecture separates functionality offered to the user from the data access functionality. This allows for future enhancements to support any data storage technology and design without changing the API provided to the developer.

The IndoWordNet API allows parallel access and updates to single or multiple language WordNets. A new design using relational database has been implemented for this purpose. This database design (IndoWordNet database) supports storage of multiple language WordNets. An effort has been made to optimize the design to reduce redundancy. Certain data common across all languages i.e. ontology information, semantic relationships, etc are stored in a separate master database and data specific to a language i.e. synsets, lexical relationships, etc are stored separately for each language in the database of respective language. The rest of the paper is organised as follows – section 2 discusses functional requirements of IWAPI, section 3 presents the architecture and design of IWAPI. The implementation details of IWAPI are presented in section 4. Section 5 presents the conclusion.

## 2 Functional Requirements

Users often have to rely on others to perform functions that he/she may not be able or permitted to do by themselves. Similarly, virtually all software has to request other software to do some things for it. To accomplish this, the asking program uses a set of standardized requests, called application programming interfaces that have been defined for the program being called upon. Developer can make requests by including calls in the code of their applications. The syntax is described in the documentation of the application being called. By providing a means for requesting program services, an API is said to grant access to or open an application.

Following information needs to be maintained for any WordNet resource. The synsets of the langauge which includes concept definition, usage examples and a set of synonym words (Miller, 1993). Each synset also belongs to a specific lexical class, namely noun, adjective, verbs and adverbs. There is also an ontology maintained and every synset maps to a specific ontology node in this hierarchy. The synsets are related through semantic relations and words are related through lexical relations. IWAPI should allow developer to access and manipulate above information. Besides the above requirement it was also felt that it should be possible to maintain additional information about the synsets i.e. an image describing a concept, pronunciations of

words in the synsets, links to websites and other resources, etc. The IWAPI should also support storage and access to such additional information. The developer based on his application requirement may also require accessing multiple language WordNets simultaneously. The IWAPI should also support this feature.

## 3 Architecture and Design

The IWAPI has two layered architecture. The upper layer is the Application layer and the lower layer is the Data layer. The class diagram of IndoWordNet API (Application layer) is shown below. The Application layer exposes the set of classes and methods which the developer will use to access and manipulate the WordNets as discussed in section 2. The Application layer does not directly access the data stored on the disk but uses Data layer for this purpose. The Data layer provides this service through a set of data classes and methods which it exposes to the Application layer. The Data layer understands the design and storage technology used to store the data i.e. relational database, flat text files, indexed files, XML etc. The Data layer is responsible for actual access and manipulation of data stored in files/database and is expected to reorganize the data in memory so that it can be exposed to the Application layer using the Data objects. This protects the Application layer from changes in storage technology or storage design. In the current implementation the Data layer accesses the data from relational

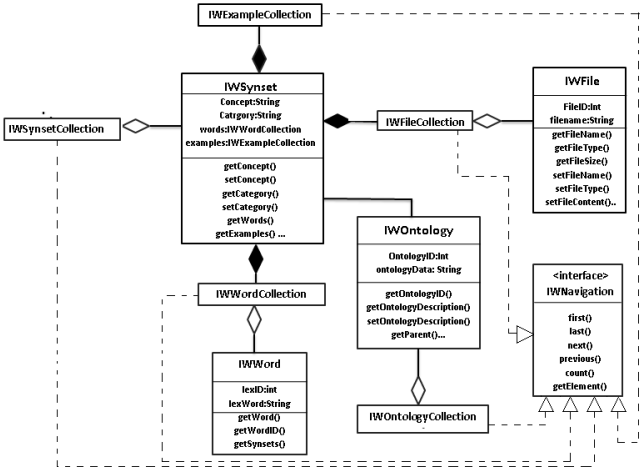


Figure 1: A simplified Class diagram of IndoWordNet API (Application Layer).

database. In future any changes in the storage method would only require a new Data layer to be implemented for that specific storage technology. Only the Application layer classes are exposed to the developer and Data layer classes are hidden so any changes will not affect the tools and software already created by the developer. The implementation maintains the classes and objects belonging to the different layers in separate libraries facilitating replacement of Data layer when desired.

Some of the important classes of Application Layer are as follows:

1. **IWAPI :** A static class that allows initialising the IndoWordNet API library for use. To use the IndoWordNet API the first thing you need to do is authenticate the user i.e. ***IWAPI.init ("username","password");***
   This class manage connectivity to language specific databases. By establishing a single connection with the master database, you can connect to multiple language specific databases using, IWAPI.getLanguageObject(IWLanguageConstants.KONKANI); where IWLanguageConstant is static class that contains all the constant names for language specific databases. It also allows developer to maintain Meta information:
   - To get/add/delete various lexical and semantic relation.
   - To get/add/delete various property values of lexical relation such as action, amount, color, direction, etc and semantic relations.
   - To manipulate domain names, grammatical categories, ontology nodes, ontology hierarchy, etc.
2. **IWLanguage:** A class that provides connection to language WordNets by using ***IWLanguage langObj = IWAPI.getLanguageObject(IWLanguageConstants.KONKANI);*** Using the langObj i.e the object of IWLanguage it allows the developer:
   - To get the total number of synsets, total number of words of a given language.
   - To get all synsets/words belonging to a domain/ category/range of Id's.
   - To create/destroy a new synset, word, domain, category, ontology, relation, etc.
   - To get words and synsets having a given semantic and lexical relations.
3. **IWSynset:** A class that represents a synset. The synset object of IWSynset class allows developer:
   - To get the concept, translated concept, transliterated concept of a given synset which is present in the database. To get usage examples of the synset.
   - To get/set the category, domain, source, concept of a given synset.
   - To add/remove examples, files and various semantic/lexical relations of a synset.
4. **IWSynsetCollection:** A class that represents a collection of synsets. It allows developer:
   - To get the size of the collection i.e. the number of synsets present in the collection, using the method, count();
   - To iterate through the collection, using getElement(); first(); next(); previous(); last(); respectively.
5. **IWWord:** A class that represents a word. The word object of IWWord class allows developer:
   - To get the id of the word, to get synsets for a given word.
   - To get various lexical relation such as antonymy relation, compounding relation, gradation relation, etc.
   - To add/remove various lexical relations for specified synset and word such as antonymy relation, compounding relation, gradation relation, etc.
6. **IWWordCollection:** A class that represents a collection of words for a synset. It allows developer:
   - To get the size of the collection i.e. the number of words present in the collection.
   - To iterate through the collection, using getElement(); first(); next(); previous(); last(); respectively.
   - It also allows deleting a word, to insert a word at a particular location, to move a word to a particular location, to change the priority of the words, etc.
7. **IWExampleCollection:** A class that represents a collection of examples for a synset. It

allows developer:

- To get the size of the collection i.e. the number of examples present in the collection.
- To iterate through the collection, using getElement(); first(); next(); previous(); last(); respectively.
- To move the current example at a specified location, to insert a new element in a collection at a specified location, to insert a new element at last position in the collection, etc.

8. **IWFile:** A class that represents files. Using the object of IWFile class it allows developer:
   - To get/set the file content, file Id, file size, file type, etc.
9. **IWOntology:** A class that represents ontology node. Each synset is mapped to an ontology node in the ontology tree. Using the object of IWOntology class the developer can
   - get/set the ontology Id, ontology data, ontology translated data, ontology transliterated data,
   - get/set the ontology description, ontology translated description, and ontology transliterated description from the database.
10. **IWOntologyCollection:** Collection of child nodes for a given onto node. Using the object of IWOntology class it allows developer:
    - To get the size of the collection i.e. the number of ontology nodes present in the collection.
    - To iterate through the collection, using getElement(); first(); next(); previous(); last(); respectively.

    Similarly we have classes such as **IWAntonymyCollection, IWGradationCollection, IWMeroHoloCollection, IWNounVerbLinkCollection,** etc.
11. **IWException:** A class that defines all the exceptions which occurs in case of error or failure.

Note: There are additional classes like IWAntonymy, IWGradation and IWMeronymyHolonymy which are the private classes used internally in the API and are hidden from the developer.

The Data layer will change depending on the storage technology but the Application layer will remain unchanged. The Data layer deals with encapsulation of the storage design. It provides a standard interface to the application layer. The Data layer supports all the operations needed to be performed on the data. Data Layer consists of the following important classes:

1. **IWDb:** A class that represents to a database/file store.
2. **IWCon:** A class that represents up an authenticated connection to a database/file store.
3. **IWStatement:** A class which contains all data manipulation functionality required by the Application layer.
4. **IWResult:** A class which returns result to the application layer i.e. synsets, collections, etc.

## 4 Implementation

Reference implementation of IndoWordNet API is done in JAVA and PHP. The size of JAVA jar file is 224 KB and the size of PHP API package is 452 KB. The IndoWordNet API classes are stored in 4 packages:

- **unigoa.indowordnet.api:** This package consists of important classes of Application Layer such as the IWAPI, IWLanguage, IWSynset, etc. as discussed earlier in section 3.
- **unigoa.indowordnet.constants:** This package consists of static classes which define all

the constants used by API and do not contain any methods.

- **unigoa.indowordnet.maintenance:** This package consists of a class which allows the developer to maintain master data.
- **unigoa.indowordnet.storage:** This package consists of classes of Data Layer such as the IWDb, IWCon, IWStatement and IWResult. They provide a standard interface to the application layer and their implementation will change depending on the storage technology.

```
1.  public class Synset {
2.  public static void main(String[] args) {
3.  String userName = "userName";
4.  String password = "password";
5.  int synsetID = 1;
6.  int category = IWGrammaticalCategoryConstants.NOUN;
7.  String concept = "Concept Definition";
8.  String word = "word";
9.  String exmple1 = "example1";
10. { IWAPI.init(userName, password);
11.  IWLanguage languageObject =
12.  IWAPI.getLanguageObject(IWLanguageConstants.KONKANI);
13.  IWSynset synsetObject =
14.  languageObject.createSynset(synsetID, concept,
     category);
     IWWord wordObject = null;
15.  try { wordObject = languageObject.getWord(word);
16.  } catch (IWException ex) {
17.  wordObject = languageObject.createWord(word);
18.  }
19.  synsetObject.addWord(wordObject);
20.  synsetObject.addExample(exmple1);
21.  String data = "";
22.  data += "Synset ID: " + synsetObject.getSynsetID();
23.  data += "\nConcept: " + synsetObject.getConcept();
24.  data += "\nCategory: " + synsetObject.getCategory();
25.  data += "\nWords: ";
26.  IWWordCollection wordCollection = synsetObject.getWords();
27.  do {
28.  IWWord wordObject1 = wordCollection.getElement();
29.  data += wordObject1.getWord() + " ";
30.  } while (wordCollection.next());
31.  data += "\nExamples: ";
32.  IWExampleCollection exampleCollection = synsetObject.getExamples();
33.  do {
34.  data += exampleCollection.getElement() + " ";
35.  } while (wordCollection.next());
36.  System.out.print(data);
37.  } catch (IWException ex) {
38.  System.err.println(ex.errorMessage());
39.  } } }
40.
```

Figure 2: Example to create a new synset using IndoWordNet API.

An example to illustrate the use of IndoWordNet API is shown above. The basic flow is as follows: Initialize the API library and authenticate the user (line no. 10). Connect to a language WordNet (Konkani WordNet line 11-12). The same method can be used repeatedly to create connections to multiple language WordNets. Each call will return an object representing connection to a language WordNet. This facilitates simultaneous access to multiple WordNets. Create a new synset using the language object (line no. 14). Add a word or an example to the new synset (line no. 19-20). Similarly, to get synset information of a given synset Id. First, create the object of the synset, and use the synset object to get synset information (line no. 22-39).

## Conclusion

The IndoWordNet API can be used to facilitate easy and rapid development of tools and other WordNet related software resources with minimal effort, with enhanced features by a developer in a very short time for any language. The ease and speed at which new tools to support research community can be developed was demonstrated during the implementation of WordNet Content Management System(CMS) which used IndoWordNet API implemented in PHP. A demo paper on WordNet Content Management System has been accepted at COLING 2012. The Konkani WordNet website is deployed using WordNet CMS.

The IndoWordNet API has been successfully used by other IndoWordNet members to develop web-based tools such as bi-lingual dictionary, multi-lingual dictionary, Lexical Relation Tool to capture various lexical relations such as antonym, gradation, etc. A Synset Management System is under development to assist creation of language specific synsets and manage their linkages to other Indian language WordNets. We expect that in future the IndoWordNet API's will be used for the development of tools and software resources by IndoWordNet members.

## References

George A. Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine Miller 1993. *Introduction to WordNet: An On-line Lexical Database.*

George A. Miller 1995. *WordNet: A Lexical Database for English*.

Joshua Bloch 2007. *How to Design a Good API and Why it Matters.*

Pushpak Bhattacharyya, *IndoWordNet, Lexical Resources Engineering Conference 2010 (LREC2010), Malta, May, 2010.*

Pushpak Bhattacharyya, Christiane Fellbaum, Piek Vossen 2010. *Principles, Construction and Application of Multilingual WordNets, Proceedings of the 5th Global Word Net Conference (Mumbai-India), 2010.*