ISSN 2320 - 2602
Volume 9 No.7, July 2020
International Journal of Advances in Computer Science and Technology
Available Online at http://www.warse.org/IJACST/static/pdf/file/ijacst24972020.pdf
https://doi.org/10.30534/ijacst/2020/24972020

Mapping Natural Language Text to Java Code

Thasneema Mullappally[1], Shibily Joseph[2]
[1]Computer Science and Engineering, GEC Palakkad, India, thasneema93@gmail.com
[2]Computer Science and Engineering, GEC Palakkad, India, shibilyj@gmail.com

ABSTRACT

Java is a class-oriented, general-purpose programming language. Mapping Natural Language Text into Java source code has become a growing demand for programmers. The proposed method generates Java source code using in-build methods in Java, from the problem statement. Java Programming Language has a hierarchical structure of Libraries, Classes, and In-build Methods. The hierarchical structure and basic syntax of the in-build methods are encoded in an Ontology. Ontology is the best way to represent data, which is machine-interpretable as well as human-readable. The Ontology created manually by using Protégé, which is an open-source ontology editor and a knowledge management system. NLP(Natural Language Processing) Techniques applied to problem statements for lexical, syntactic, and semantic analysis. The SPARQL query can be used to retrieve appropriate in-build methods from the Ontology. Based on the queried information from the Ontology, sequences of code lines can be generated.

Key words: NLP, Protégé, Ontology, SPARQL

1. INTRODUCTION

Sometimes, the programmers may face a situation that, they knows the logic of the program, but doesn't know the proper usage and syntax of the programming language constructs. Mapping Natural Language Text to Java code is a system that automatically transform a given statement to Java code from in-build methods in Java. The statement or natural language text should be a short description of code to be generated. Suppose a Programmer wants to connect a client to server in his application code, he can just write "connect to server". The system suggest Java code for connecting to a server. The programmer can reuse the generated code with some alterations such as changing variable name, inputting methods etc. This system reduce the programmers effort and complexity while writing a complete code, and can develop applications after making appropriate adjustments or alteration in the generated code.

Java is a powerful general-purpose programming language[1]. There are millions of applications across multiple platforms that use java programming language such as Desktop GUI Applications, Mobile Applications, Embedded Systems, Web Application, Web Servers and application Servers, Enterprise Applications, Scientific Applications etc. According to Oracle, the company that owns Java, Java runs on 3 billion devices worldwide, which makes Java one of the most popular programming languages. It contains a large number of in-build methods under different classes. These classes are organized under Java library packages. This hierarchical structure of Java programming language can be encoded in an Ontology.

The proposed method is an ontology based work. An ontology is a formal description of knowledge within a domain, that represent knowledge as a set of concepts and the relationships that hold between them [2]. To enable such a description, ontology provides formally specifying components such as individuals (instances of objects), classes, attributes and relations as well as restrictions, rules and axioms. The Knowledge represented in ontology is both machine-interpretable as well human readable. One of the main features of ontologies is that, by having the essential relationships between concepts built into them, they enable automated reasoning about data [3]. So the System can retrieve most appropriate information from the Ontology. In addition, ontologies provide a more coherent and easy navigation as users move from one concept to another in the ontology structure. Another valuable feature is that ontologies are easy to extend as relationships and concept matching are easy to add to existing ontologies. As a results, ontologies can be modified without effecting the System process. Ontologies also provide the means to represent any data formats, including unstructured, semi-structured or structured data, enabling smoother data integration, easier concept and text mining, and data-driven analysis. The Ontologies have some limitations too. One such limitation is the available property constructs. The most recent version of the Web Ontology Language (OWL2) has a somewhat limited set of property constructs. Another limitation comes from the way OWL employs constraints. The data imported from a new source into the RDF triple-store would be structurally inconsistent with the constraints set using OWL.

An ontology for java programming language can be build manually by using Protégé. The Protégé is a free, open source ontology editor and framework for building

intelligent systems. This manually created Java ontology can be used for Java code sequence generation. Since Java language contains a large number of classes and in-build methods, Ontology construction for Java language is a labor intensive and complex process. But once the ontology is built the major task is complete. It can easily used for code sequence generation.

2. RELATED WORK

The Natural Language can be used to define complex computations tasks. There are different approaches for automatically mapping natural language (NL) to executable code. Some approaches assume fixed code templates [2], that generate only parts of a method with a predefined structure [4], some consider a fixed context can generate the body of the same method within a single fixed class [5] and some doesn't consider any context at all, that generate code tokens from the text alone as in [6].

There are also different approaches for code generation. Many works are based on neural network model. The [7] is a Encoder-Decoder model with a supervised copy mechanism for Java code generation, that also consider the programmatic context. A neural architecture powered by a probabilistic grammar model is used in [8] for general purpose programming language code generation. The [5] is also a general framework for general purpose code generation. Some approaches use language and task specific rules as in [9], [10] for code generation. Finally, there are probabilistic models for source code generation as in [11], [12]. The most relevant work [13], which uses a factorized model to measure semantic relatedness between NL and Abstract Syntax Trees (ASTs) for code retrieval.

There is also reverse approaches, that is to generate Natural Language summaries from source code has also been explored in [14] and [6].

3. PROBLEM DEFINITION

The Problem can be stated as:
Design and Develop a System that transform Natural Language Text to Java Code which use in-build methods present in Java Programming Language.

Mathematically, Given an NL description ***N*** , the task is to generate the Java code sequence ***J,*** which uses in-build methods ***i***, with the help of an Ontology ***O*** in Java programming language domain.

4. SYSTEM ARCHITECTURE

This section gives detail explanation about the architecture of the System and working as shown in Figure 1. NL Text preprocessing, Ontology Creation, SPRQL Query Generation, Method Syntax Extraction, and Code Sequence Generation are the major step of this Architecture.

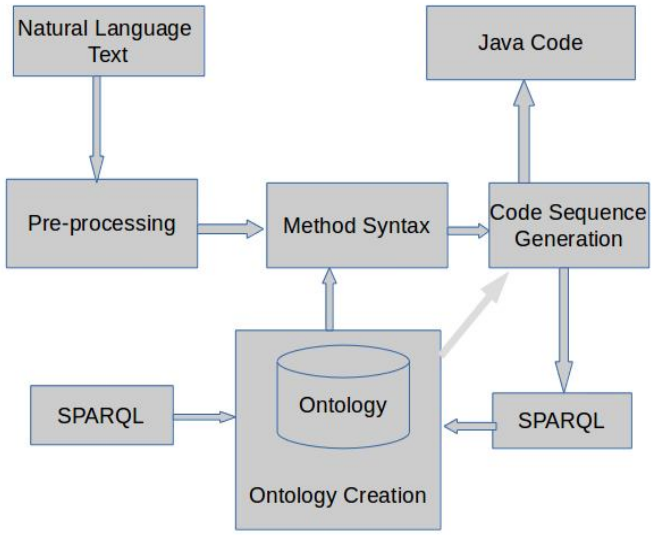

Figure 1: System Architecture

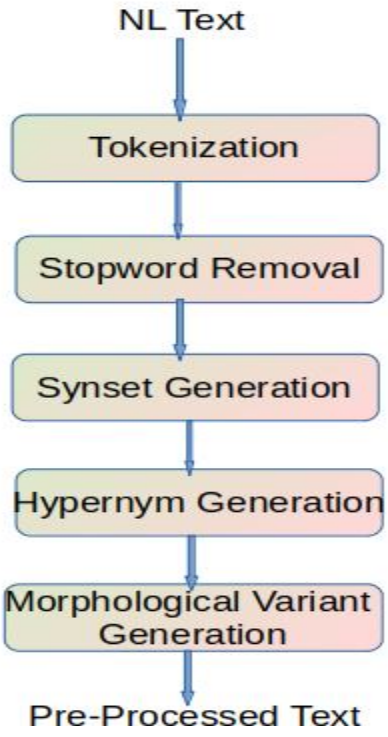

Figure 2: NL Text Pre-rocessing

4.1. Text Pre-Processing

Initially, the system needs to understand the Natural Lan guage(NL) Text. So it analyses text both syntactically and semantically using NLTK module [2]. The Figure 2 shows

the major pre-processing tasks, that are done during this stage.

*1)***Tokenization:** The Natural Language Text is tokenized by word Tokenizer. Word Tokenizer generates a sequence of words using NLTK module, which is implemented in python.

2) **Stopword Removal**: The Natural Language Text that can be processed by this System should be in English. The stopwords such as ”the”, ”is”, ”on” etc doesn’t play any role to identify correct in-build Java method from Ontology. So the stopwords can be eliminated by using NLTK.

3) **Synset Generation:** The Synset of every tokenized word is generated and added to a list. Because the System needs to identify ”gets” and ”returns” are similar.

4) **Hypernym Generation:** Hypernym of every word is also added to the list using NLTK module. Since whenever the user enter text as ”integer” or ”digit”, the system needs to identify it as ”number”.

5) **Morphological Variant Generation:** The morphological variants of each word are generated and added to the same list. For a plural noun words its singular form added and for a singular noun its plural added. For a verbs, its different form such as past, present etc also added to list. The system need to treat ”get”, ”gets”, ”getting” etc are same.

4.2 Ontology Creation

The major part of this System is a Java Ontology. Java ontology contains Knowledge about Java programming language constructs. This Java ontology is used later by the System for information retrieval. Java programming language has a Hierarchical structure. That is Java contains libraries. Each library contains a large number of classes. Each class contains a large number of in-build methods. The Figure 3 shows the Hierarchical structure of Java programming language [2].

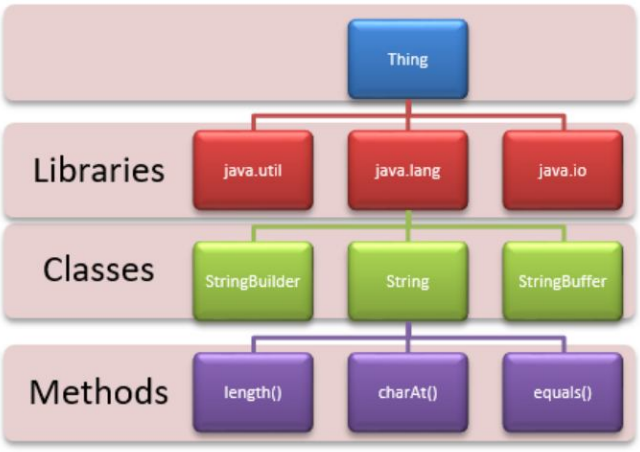

Figure 3: Hierarchical Structure of Java Language

Java Ontology contains following elements:

1) *Hierarchical Structure of Java*
2) *Class Constructor:* if class has constructor,then default class constructor syntax encoded
3) *Method Description:* A short natural language text describing what the method do.
4) *Method Syntax:* Syntax of in-build method

The Figure 4 shows a small part of ontology, which represent in-build method syntax and method description.

```
−<!--
    http://www.semanticweb.org/mtechcl/ontologies/2020/0/untitled-ontology-3#String.length
 -->
−<owl:Class rdf:about="http://www.semanticweb.org/mtechcl/ontologies/2020/0/untitled-ontology-3#String.length">
   <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/mtechcl/ontologies/2020/0/untitled-ontology-3#String"/>
  −<rdfs:comment>
     This method returns the length of this string or character sequences
   </rdfs:comment>
   <rdfs:isDefinedBy>public int length()</rdfs:isDefinedBy>
 </owl:Class>
```

Figure 4. Description of a Method

Protégé is a free, open-source ontology editor and framework, that is used to build Java ontology [15]. The class concepts and relationship among them can be easily insert in to ontology using Protégé. It generate syntactically correct ontology and save the ontology as *.rdf* format.

4.3 SPARQL Query Generation

SPARQL (SPARQL Protocol and RDF Query Language) is an RDF query language, that is, a semantic query language for databases. SPARQL has ablility to retrieve and manipulate data stored in Resource Description Framework (RDF) format. SPARQL Query generated in two phases of the System.
1) To get Method Syntax
2) To get Class constructor Syntax and Java Library at the time of code sequence generation

For Example below SPARQL query extract all java class of *java.util* and its corresponding comment.

```
SELECT  ?v ?c
WHERE {
        ?v rdf s : subClassOf  ‘java.util’.
        ?v rdf s : comment ?c.
}
```

4.4 Method Syntax Extraction

Based on the Pre-processed Natural Language Text, queried the Java ontology to in-build method syntax corresponds to NL Text. Method syntax extraction is used to identify what are the inputs and outputs needed for this in-build method.

4.5 Code Sequence Generation

In this phase Java code sequences is generated based on identified in-build method from method Syntax Extraction phase. Inorder to generate Java code sequence, some additional information also needed, the are, in which class the method resides, corresponding java library and if class has constructor, then its constructor syntax. SPARQL Query again used this phase to get these information from ontology. After extracting these information, the System generate Java code sequences through straight-line programming.

5. RESULT AND ANALYSIS

The result of the system through different stage is given below:

1) The input to the system is a Natural Language Text indicating code to be generated. This NL Text is entered by a programmer or user.
Example:
• *System:* "Enter Natural Language Text description of code to be generated"
• *User:* "Get length of a String

2) The next step produces a list of words after pre-processing of user input. This list of words contains major words of input text, its different morphological forms, syntactic and semantic variants etc.
Example: For the above NL Text, the word list contains get, gets, return, returns, length, distance, long, string, sequence, sequences etc.

3) This step is most challenging one. Based on the generated word list query the ontology, identify most matching in-build method syntax from ontology and extract it. Java programming language contains in-build method having same name indifferent classes. These methods have functionality that is almost similar. For example, the in-build method *read()* is present under the following classes. *FileReader*, *InputStream*, *FileInputStream* etc. The *read()* method in each class do same or different function. So identifying which *read()* method user wants is a challenging task. So the System suggest three Java code sequences for a particular NL Text. The user can select most appropriate Java code from the suggested code sequences.

4) After querying and retrieving some more information such as class constructor, library etc. Java code sequences generated by straight-line programming. Python is a high level programming language, which allow string processing quit easy.
Example: The Java code sequence corresponds to the NL Text "get length of a string" is as shown in Figure 5 .

The systems suggest three code sequences in which one code is most appropriate. Most probably it is the first code sequence. The system provides a user friendly GUI as shown in Figure 6.

```
import java.util.*;
import java.lang.String;
public class ClassName {
public static void main(String[] args){
  System.out.println("Enter the string");
  String St;
  Scanner sc = new Scanner(System.in);
  if(sc.hasNext())
     St=sc.next();
  String Str= new String(St);
  int OutVar;
  OutVar = Str.length();
  System.out.println(OutVar);
  }
}
```

Figure 5. Java code sequence Output Exampl

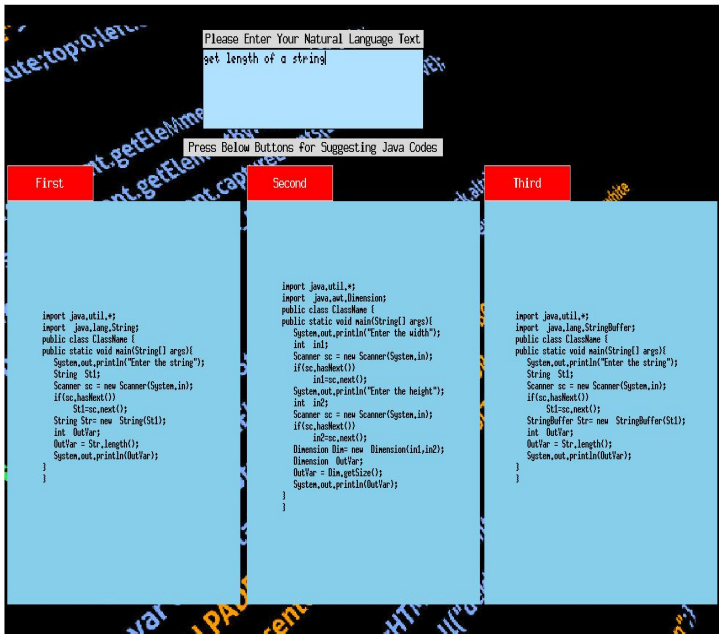

Figure 6. GUI of the System

6. CONCLUSION

Mapping Natural Language Text to Java Code is very helpful for Java programmers, who may stuck because of unaware of proper programming language constructs. This System automatically generates Java code by using Java Ontology. It can only generate Java code using in-build methods present in Java. Java Ontology, is main part of the System, which is

a Knowledge base in Java domain, contains knowledge about Java in-build methods. The System uses a manually created Java Ontology, which is a time consuming process. The first phase of the System is some of the Natural Language Processing (NLP) techniques, which are done using NLTK module in Python. SPARQL Query Language is used for the information retrieval phase from Java Ontology. Protégé , is another tool used for the creation of Java Ontology which is a major contribution of this work.

7. FUTURE SCOPE

In the future, I plan to modify or include the following advancements to the System.

• Some classes are inherits some methods from the other classes. This information doesn't included in Java Ontology. So it is not possible to generate a code sequence, that use inherited in-build methods of a class. In future needs to add these information to Java Ontology to produce a more useful Java code sequences.
• In the Preprocessing step, need to try some other existing NLP models to find semantic similarity of two texts and compare the performance of the System.
• Since manual creation of Ontology is a time consuming process, need to try to create Java Ontology automatically from Java programming language documentation or from some other source.

REFERENCES

1. K. Arnold, J. Gosling, D. Holmes, and D. Holmes, ***The Java programming language,*** vol. 2. Addison-wesley Reading, 2000.
2. A. Kulkarni, S. Karandikar, P. Bamhore, S. Gawade, and D. Medhane, ***Computational intelligence model for code generation from natural language problem statement,*** *Fourth International Conference on Computing Communication Control and Automation (ICCUBEA)*, pp. 1–6, IEEE, 2018.
3. Y. Shu, Y. HaiLun, Y. XiangRun, and W. Ye, ***An automated method for constructing ontology,*** *7th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, pp. 538–541, IEEE, 2016.
4. I. Beltagy and C. Quirk, ***Improved semantic parsers for if-then statements,*** in Proceedings of the *54th Annual Meeting of the Association for Computational Linguistics* (Volume 1: Long Papers), pp. 726–736, 2016.
5. W. Ling, E. Grefenstette, K. M. Hermann, T. Kočiskỳ, A. Senior, F. Wang, and P. Blunsom, ***Latent predictor networks for code generation,*** *arXiv preprint* arXiv:1603.06744, 2016.
6. Y. Oda, H. Fudaba, G. Neubig, H. Hata, S. Sakti, T. Toda, and S. Nakamura, ***Learning to generate pseudocode from source code using statistical machine translation,*** in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 574–584, IEEE, 2015.
7. S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, ***Mapping language to code in programmatic context,*** *arXiv preprint arXiv:*1808.09588, 2018.
8. P. Yin and G. Neubig, ***A syntactic neural model for general-purpose code generation,*** *arXiv preprint arXiv:1704.01696*, 2017.
9. T. Lei, F. Long, R. Barzilay, and M. Rinard, ***From natural language specifications to program input parsers,*** *in Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1294–1303, 2013.
10. M. Raghothaman, Y. Wei, and Y. Hamadi, ***Swim: Synthesizing what I mean-code search and idiomatic snippet synthesis,*** in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pp. 357–367, IEEE, 2016.
11. C. Maddison and D. Tarlow, ***Structured generative models of natural source code,*** in *International Conference on Machine Learning*, pp. 649–657, 2014.
12. T. T. Nguyen, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, ***A statistical semantic language model for source code***, *in Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, pp. 532–542, 2013.
13. M. Allamanis, D. Tarlow, A. Gordon, and Y. Wei, ***Bimodal modelling of source code and natural language,*** *in International conference on machine learning*, pp. 2123–2132, 2015.
14. S. Iyer, I. Konstas, A. Cheung, and L. Zettlemoyer, ***Summarizing source code using a neural attention model,*** *in Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics* (Volume 1: Long Papers), pp. 2073–2083, 2016.
15. N. F. Noy, M. Sintek, S. Decker, M. Crubézy, R. W. Fergerson, and M. A. Musen, ***Creating semantic web contents with protege-2000,*** *IEEE intelligent systems*, vol. 16, no. 2, pp. 60–71, 2001.