Volume 5, Issue 3, March 2015 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Conversion of Natural Language into Java Programe

Rajashri S. Chaudhari[1], Dipali A. Chaudhari[2], Shubjangi B. Patil[3], Leena R. Waghulde[4]
[1, 2, 3] B.E. Computer, [4] Asst. Professor
KCES's COEIT, Computer Department, North Maharashtra University,
Jalgaon, Maharashtra, India

Abstract—There is a learning process for everyone who expects to be a decent programmer in any programming language. To write any application one must have to understand the basics of any programming language. The process of understanding the basics involves designing the algorithms for a simple problems and writing the programs for those. The syntaxes are playing important role in it. The proposed paper gives an idea of designing an application that converts these algorithms into application programs.

Keywords— NLP, Algorithm, Application, Programme

I. INTRODUCTION

"Programming is an art of implementing a process into computer statement." Yes it is true that whenever you are writing down any application it is set of small activities to be converted into computer statement. Each type of activity has some defined syntaxes in a particular programming language. Every program is set of such syntaxes statement. It is very easy to define the process in the form which we used to understand and we are very much happy to use the specific word for a specific type of any activity. However the same word cannot be used in writing the computer program in a particular programming language. This makes the programming very difficult. Even if we understand the process we may not be able to convert it into computer program at the initial stage. Many programmers at this point decides to quit the programming and starts to think that programming is not their cup of tea. This is where the focus has to be given and shall allow to write the program in natural language and shall be converted to its equivalent language program. This can be done using a small trick presented within the Natural language processing. The proposed process is exactly describes as to how to convert natural language algorithm into the Application program for a specific programming language.

Any programming language in computer shall consider some important aspects of programming like the defining the variable using a specific type of value to store the data in it, mostly known as Data Type, general syntax to read the data into the variable, displaying data or the validating the condition or looping through the statements. Where as in NLP we used to write the statement in our likely language as well as we used to use the words familiar to us. Thus it is necessary to consider the specified aspects of programming language and convert the NLP written algorithm into programming language based programs. The process shall be more simpler and easier to make the new programmer more comfortable with the language and shall explain various basics of the language with ease[1].

This paper makes three main contributions. First, we demonstrate that there exists a theoretical model that describes most, if not all, NLP approaches; second, we describe our proposed model to language used by basic programmer and show that they enable the programmer to describe the process of writing programs. Third, we introduce the concept of Application that explains translation process in which the efficiency of the generated code benefits from the data given as input to the learning algorithms. Thus, the programmer spends his time designing his models instead of worrying about the low level details of writing efficient learning based programs that have been abstracted away.

II. NATURAL USER INTERFACE

The goal of the Natural Java interface is to allow programmers to write computer programs by expressing each command in natural language. For example, a user might ask the system to "create a for loop that iterates from 1 to 10." This form of interaction allows the programmer to give instructions without having to know the exact syntax required by the programming language[2].

III. MOTIVATION AND BACKGROUND

Natural language (NL) interfaces can be plagued with two types of problems: natural language specifications can be ambiguous and incomplete, and natural language processing can be fragile because complete NL understanding is still beyond the state of the art. We addressed the first problem by limiting the role of inference in our system. Natural Java recognizes NL commands that, while very similar to actual programming constructs, are expressed in English. This level of specification is relatively well defined, yet general enough that the programmer can focus on programming rather than syntax. The interface can detect when a command is incomplete (e.g., the terminating condition of a loop is missing) and prompt the user, but the role of inference in Natural Java is mainly limited to the disambiguation of general verbs (e.g., "add" can refer to arithmetic or insertion)[2]. We addressed the second problem of fragile natural language processing by

using information extraction technology supported by a partial parser. Partial parsers are typically more robust and flexible than full parsers, which try to generate a complete parse tree for each sentence. Full parsers often fail on sentences that are ill-constructed or ungrammatical. Partial parsers are more robust because they do not have to generate a complete parse structure, but instead generate a flat syntactic representation of sentence fragments. A few NL interfaces have been previously developed for programming. Perhaps the biggest difference between Natural Java and previous systems is that Natural Java allows users to generate and manipulate source code in a real programming language using an AST.

IV. EXISTING MODEL OF NLP

"Natural language processing" here refers to the use and ability of systems to process sentences in a natural language such as English, rather than in a specialized artificial computer language such as C++. The systems of real interest here are digital computers of the type we think of as personal computers and mainframes (and not digital computers in the sense in which "we are all digital computers," if this is even true). Of course humans can process natural languages, but for us the question is whether digital computers can or ever will process natural languages.

Unfortunately there is some confusion in the use of terms, and we need to get straight on this before proceeding. First of all, occasionally the phrase "natural language" is used not for actual languages as they are used in ordinary discourse, such as our actual use of English to communicate in everyday life, but for a more restricted subset of such a human language, one purged of constructions and ambiguities that computers could not sort out. Hence one writer states that "human languages allow anomalies that natural languages cannot allow." There may be a need for such a language, but a natural language restricted in this way is artificial, not natural. I do not use the phrase "natural language" in this restricted sense of an artificial natural language. When I use the phrase, I mean human language in all its messiness and varied use.

Second, the phrase "natural language processing" is not always used in the same way. There is a broad sense and a narrow sense. The phrase sometimes is taken broadly to include signal processing or speech recognition, context reference issues, and discourse planning and generation, as well as syntactic and semantic analysis and. At other times the phrase is used more narrowly to include only syntactic and semantic analysis and processing.

Third, even if this confusion is overcome, the phrase "natural language processing" may or may not be taken as synonymous with "natural language understanding." "Processing" most naturally is used for both interpretation and generation, while one would think "understanding" is better used for only the interpretation part. To me, to say that a system is capable of natural language understanding does not imply that the system can generate natural language, only that it can interpret natural language. To say that the system can process natural language allows for both understanding (interpretation) and generation (production). But the phrase "natural language understanding" seems used by some authors as synonymous with "natural language processing," and on this use includes interpretation and generation. In this paper we use the phrase natural language processing, but keep in mind we are mostly just discussing interpretation rather than generation [3].

Fourth, another problem is that the phrase "natural language understanding" is sometimes used in the sense of research into how natural languages work and the attempt to develop a computational model of this working, with "natural language processing" referring to a system interpreting and generating natural languages. This use of the phrase alludes to two distinct goals of this field of research. One goal is to understand how natural language processing works; here "natural language understanding" is a human endeavor to understand natural language processing, whoever does the processing. Another goal is to get a computer to process natural languages, and of course in this attempt to build a natural language processor fulfilling the first goal of understanding how a processor works could be useful. Another way to distinguish these two goals is to note that in seeking to develop a computational model of natural language processing (or understanding), one must distinguish between the scientific goal and the technical goal. In this attempt to develop a computational model, there is the scientific goal or motivation of understanding natural language comprehension and production for its own sake. There is also usually associated the technical goal of getting a computer to process natural language sentences. Allen at first seems to distinguish natural language understanding as a human scientific endeavor from natural language processing as something that endeavor is trying to investigate and model, but then at times even he uses "natural language understanding" in a way that requires one to figure out which sense is intended[4].

There is a fifth confusion to sort out, and this is that the term "understanding" is loaded with implications or connotations that aren't present with "processing," though most users of the phrase "natural language understanding" seem either unaware or unconcerned about these loaded connotations. Humans are of course able to process and understand natural languages, but the real interest in natural language processing here is in whether a computer can or will be able to do it. Because of the connotations of the term "understanding," it's use in the context of computer processing should be qualified or explained. Searle, for example, claims that digital computers such as PCs and mainframes, as we currently know them, cannot understanding anything at all, and no future such digital computer will ever be able to understand anything by virtue of computation alone. Others disagree. In view of this controversy over whether digital computers of the type under consideration can ever understand anything, use of the phrase "natural language understanding" seems to slant the issue unless one qualifies the use of the phrase "understanding." But many authors use the phrase "understanding" as if the term had no controversial history with respect to computers. The term "processing" is perhaps preferable to "understanding" in this context, but "understanding" has a history here and we are not advocating we discontinue use of the term. Certainly in this paper if we use the terms "understanding" or "knowledge" metaphorically with reference to computers, we imply nothing about whether they can or will ever really understand or know in any philosophically interesting sense [5].

V. UNDERSTANDING COMMANDS USING INFORMATION EXTRACTION

Information extraction (IE) is a form of natural language processing that involves extracting predefined types of information from natural language text. The goal is to identify information that is relevant to the task at hand while ignoring irrelevant information. Information extraction systems have been built for a variety of domains, including Latin American terrorism ,joint ventures , microelectronics , job postings , rental ads , and seminar announcements . For the Natural Java interface, we used IE techniques to extract information related to Java programming constructs from the user's input. The natural language engine used by Natural Java is a partial parser called Sundance, which was developed at the University of Utah. Sundance generates a flat syntactic representation of sentences and also can activate and instantiate pattern-based templates, or case frames. For the Natural Java task, we manually designed 400 case frames to extract information about relevant programming constructs. As an example, consider the sentence "Create a for loop that iterates from 1 to 10." Sundance begins by deriving a partial parse for this sentence, which involves part-of-speech disambiguation, syntactic bracketing, clause segmentation, and syntactic role assignment. Sundance then instantiates all active case frames to extract information from the sentence. The case frames represent local linguistic expressions revolving around verbs and nouns. Each case frame has a trigger word and an activating function that determines when it is applicable. For example, a case frame might be triggered by the word "iterates" when it appears as an active verb form. A case frame also has a type, which represents its general concept, and an arbitrary number of slots that extract information from local syntactic constituents [6].

VI. DESCRIPTIVE NATURAL LANGUAGE PROGRAMMING

When storytellers speak fairy tales, they first describe the fantasy world – its characters, places, and situations – and then relate how events unfold in this world. Programming, resembling storytelling, can likewise be distinguished into the complementary tasks of description and proceduralization. While this paper tackles primarily the basics of building procedures out of steps and loops, it would be fruitful to also contextualize procedural rendition by discussing the architecture of the descriptive world that procedures animate. Among the various paradigms for computer programming – such as logical, declarative, procedural, functional, object-oriented, and agent-oriented – the object-oriented and agent-oriented formats most closely embody human storytelling intuition. Consider the task of programming a MUD2 world by natural language description, and the sentence there is a bar with a bartender who makes drinks [6]. Here, bar is an instance of the object class bar, and bartender is an instance of the agent (a class with methods) class bartender, with the capability make Drink(drink). Generalizing from this example, characters are reified as agent classes, things and places become object classes, and character capabilities become class methods. A theory of programmatic semantics for descriptive natural language programming is presented in [6].

VII. PROCEDURAL NATURAL LANGUAGE PROGRAMMING

Procedural Natural Language Programming of computer programs following the procedural paradigm, starting with a natural language text. For example, starting with the natural language text on the left side of figure 2, we would ideally like to generate a computer program as the one shown on the right side of the figure3. While this is still a long term goal, in this section we show how we can automatically generate computer program skeletons that can be used as a starting point for creating procedural computer programs. Specifically, we focus on the description of three main components of a system for natural language procedural programming:

1. The step finder, which has the role of identifying in a natural language text the action statements to be converted into programming language statements.
2. The loop finder, which identifies the natural language structures that indicate repetition.
3. Finally, the comment identification components, which identifies the descriptive statements that can be turned into program comments.

Starting with a natural language text, the system is first analyzing the text with the goal of breaking it down into steps that will represent action statements in the output program. Next, each step is run through the comment identification component, which will mark the statements according to their descriptive role. Finally, for those steps that are not marked as comments, the system is checking if a step consists of a repetitive statement, in which case a loop statement is produced using the corresponding loop variable.

VIII. THE STEP FINDER

The role of this component is to read an input natural language text and break it down into steps that can be turned into programming statements. For instance, starting with the natural language text you should count how many times each number is generated and write these counts out to the screen two main steps should be identified:

(1) [count how many times each number is generated], and
(2) [Write these counts out to the screen].

First, the text is pre-processed, i.e. tokenized and part-of-speech tagged using Brill's tagger [2]. Some language patterns specific to program descriptions are also identified at this stage, including phrases such as write a program, create an apple, etc., which are not necessarily intended as action statements to be included in a program, but rather as general directives given to the programmer. Next, steps are identified as statements containing one verb in the active voice. We are therefore identifying all verbs that could be potentially turned into program functions, such as e.g. read,

write, count. We attempt to find the boundaries of these steps: a new step will start either at the beginning of a new sentence, or whenever a new verb in the active voice is found (typically in a subordinate clause). Finally, the object of each action is identified, consisting of the direct object of the active voice verb previously found, if such a direct object exists. We use a shallow parser to find the noun phrase that plays the role of a direct object, and then identify the head of this noun phrase as the object of the corresponding action. The output of the step finder process is therefore a series of natural language statements that are likely to correspond to programming statements, each of them with their corresponding action that can be turned into a program function (as represented by the active voice verb), and the corresponding action object that can be turned into a function parameter (as represented by the direct object). As a convention, we use both the verb and the direct object to generate a function name. For example, the verb write with the parameter number will generate the function call write Number (number).

IX. THE LOOP FINDER

An important property of any program statement is the number of times the statement should be executed. For instance, the requirement to generate 10000 random numbers, implies that the resulting action statement of [generate random numbers] should be repeated 10000 times. The role of the loop finder component is to identify such natural language structures that indicate repetitive statements. The input to this process consists of steps, fed one at a time, from the series of steps identified by the step finder process, together with their corresponding actions and parameters. The output is an indication of whether the current action should be repeated or not, together with information about the loop variable and/or the number of times the action should be repeated. First, we seek explicit markers of repetition, such as each X, every X, all X. If such a noun phrase is found, then we look for the head of the phrase, which will be stored as the loop variable corresponding to the step that is currently processed. For example, starting with the statement write all anagrams occurring in the list, we identify all anagrams as a phrase indicating repetition, and anagram as the loop variable. If an explicit indicator of repetition is not found, then we look for plural nouns as other potential indicators of repetition. Specifically, we seek plural nouns that are the head of their corresponding noun phrase. For instance, the statement read the values contains one plural noun (values) that is the head of its corresponding noun phrase, which is thus selected as an indicator of repetition, and it is also stored as the loop variable for this step. Note however that a statement such as write the number of integers will not be marked as repetitive, since the plural noun integers is not the head of a noun phrase, but a modifier. In addition to the loop variable, we also seek an indication of how many times the loop should be repeated – if such information is available. This information is usually furnished as a number that modifies the loop variable, and we thus look for words labelled with a cardinal part-of-speech tag. For instance, in the example generate 10000 random numbers, we first identify numbers as an indicator of repetition (noun plural), and then find 10000 as the number of times this loop should be repeated. Both the loop variable and the loop count are stored together with the step information. Finally, another important role of the loop finder component is the unification process, which seeks to combine several repetitive statements under a common loop structure, if they are linked by the same loop variable [5]. For example, the actions [generate numbers] and [count numbers] will be both identified as repetitive statements with a common loop variable number, and thus they will be grouped together under the same loop structure.

X. COMMENT IDENTIFICATION

Although not playing a role in the execution of a program, comments are an important part of any computer program, as they provide detailed information on the various programming statements. The comment identification step has the role of identifying those statements in the input natural language text that have a descriptive role, i.e. they provide additional specifications on the statements that will be executed by the program. Starting with a step as identified in the step finding stage, we look for phrases that could indicate a descriptive role of the step. Specifically, we seek the following natural language constructs: (1) Sentences preceded by one of the expressions for example, for instance, as an example, which indicate that the sentence that follows provides an example of the expected behaviour of the program. (2) Statements including a modal verb in a conditional form, such as should, would, might, which are also indicators of expected behaviour. (3) Statements with a verb in the passive voice, if this is the only verb in the statement4. (4) Finally, statements indicating assumptions, consisting of sentences that start with a verb like assume, note, etc. All the steps found to match one of these conditions are marked as comments, and thus no attempt will be made to turn them into programming statements. An example of a step that will be turned into a comment is *For instance, 23 is an odd number,* which is a statement that has the role of illustrating the expected behaviour of the program rather than asking for a specific action, and thus it is marked as a comment. The output of the comment identification process is therefore a flag associated with each step, indicating whether the step can play the role of a comment [5]. Note that although all steps, as identified by the step finding process, can play the role of informative comments in addition to the programming statements they generate, only those steps that are not explicitly marked as comments by the comment identification process can be turned into programming statements. In fact, the current system implementation will list all the steps in a comment section , but it will not attempt to turn any of the steps marked as "comments" into programming statements.

XI. CONCLUSION AND FUTURE WORK

We emphasized the benefits of representing the information found in natural language algorithm facilitate automated Java source code analyses. We discussed how the Natural language to Java converter is able to generate a Java Program using information found in algorithm. And finally, we compared our Natural language to Java converter with other

converters that are currently published, and pointed out that it is the first to be explicitly designed for generating a Java program that is to be used within Java applications. From this project work we conclude that the Natural Language Compiler translating the Natural language algorithms into Java language programs. This software will provide a great deal of help to new programmers. However it will be very useful for the professional programmers to convert some simple algorithms into JAVA programs.

In future the system shall be developed to work with the algorithms written with the use of advanced features of Java programming language. In short, In this software as user demands it can be further implement it for all the 'Java' syntax. This software can be modified further for control statements like 'Array', 'Pointers', 'Structure' etc. In short, we can modify this software as user demands. We can further implement it for all the 'Java' syntax. Future work is specifically needed in following area, marking-up more of the information provided by algorithm.

ACKNOWLEDGMENT

The success of this paper needed co-operation & guidance of number of people. We, therefore consider it as our prime duty to thank all those who had helped me through this venture. It gives me immense pleasure to express gratitude to Guide Prof. Leena R. Waghulde, who provided us constructive and positive feedback during the preparation of this paper. We express our sincere thanks to all other staff members of Computer Engineering Department for their kind co-operation. Lastly, We express our ineptness to all those who directly or indirectly helped us in making the completion of this paper possible.

REFERENCES

[1] Ballard, B, and Bierman, "A Programming in natural language: "NLC" as a prototype". *In Proceedings of the 1979 annual conference of ACM/CSC-ER (1979).*

[2] David Price, Ellen Riloff, Joseph Zachary, Brandon Harvey, "NaturalJava: A Natural Language Interface for Programming in Java",

[3] Califf, M. E. Relational Learning Techniques for , "Natural Language Information Extraction", Ph.D. Dissertation,Tech. Rept. AI98-276, Artificial Intelligence Laboratory,The University of Texas at Austin, 1998

[4] Freitag, D. Multistrategy, " Learning for Information Extraction", In Proceedings of the Fifteenth International Conference on Machine Learning , 1998.

[5] H.Liu, H, and Lieberman, H. Liu, " Programmatic semantics for natural language interfaces", *In Proceedings of the ACM Conference on Human Factors in Computing Systems (CHI-2005) (Portland, OR, 2005).*

[6] Rada Mihalcea1, Hugo Liu2, and Henry Lieberman2, " NLP (Natural Language Processing) for NLP (Natural Language Programming)", A. Gelbukh (Ed.): CICLing 2006, LNCS 3878, pp. 319–330, 2006.

