Journal of Computing and Information Technology - CIT 15, 2007, 1, 25–32
doi:10.2498/cit.1000755

A Natural Language Architecture

Adesina Simon Sodiya

Computer Science Department, University of Agriculture, Abeokuta, Ogun State, Nigeria

Natural languages are the latest generation of programming languages, which require processing real human natural expressions. Over the years, several groups or researchers have been trying to develop widely accepted natural languages based on Artificial Intelligence (AI). But, no true natural language has been developed. The goal of this work is to design a natural language preprocessing architecture that identifies and accepts programming instructions or sentences in their natural forms and generates equivalent codes in the base high-level language. The new programming language platform, called H++, translates and processes real human natural expressions. Using Visual Basic 6.0 as the basic high-level programming language, the implementation resulted in an interactive and easy to use natural language platform.

Keywords: programming language, natural language, artificial intelligence (AI), instructions.

1. Introduction

A high-level programming language is a programming language that is user-friendly, to some extent platform-independent, and abstract from low-level computer processor operations (Bunch and Hellemans, 2004). It is therefore evident that a user-friendly programming language must be easy to use. High-level programming languages were designed to be easy to use by making them closer to human language than to the machines languages. But, most high-level programming languages still require coding, though these codes are close to human expressions.

Coding in most programming languages requires writing lines of instructions, even for a very simple problem. The object-oriented programming languages have been developed to also, among other benefits, reduce the lines of codes. However, in most programming languages, the simplest problem definition would require a minimum of 4 to 5 lines of codes. A huge programming task at times requires good programming skills. In some cases, programming in high-level programming language is complex and less interesting because of the requirement for core knowledge of the language syntax. This is why there is much interest in the design and processing of natural language. Natural Language Processing (NLP) requires analyzing, understanding and generating languages that human beings use naturally, so that eventually you will be able to address the computer as if you were addressing another person. The goal of NLP is not easy to reach as it requires detailed syntax and semantic analysis and representations.

The objective of this work is to design a natural language platform that brings programming in high-level languages much closer to human languages. It is geared towards given natural language instructions to computer systems. The idea is to remove or reduce the jargons of coding so as to make programming an easy task for skilled or less skilled programmers or computing professionals. The new programming platform is called H++ (pronounced Hplusplus), which means a higher high-level programming language. H++ is a unique programming environment that will make the act of programming faster, easier and friendlier.

2. Related Work

The third and fourth generation languages are generally less technical, less flexible, slower and more user-friendly than the previous ones (Williams et al., 1999) and this has brought the need for another level of programming language. Though these languages use English-like expressions to write instructions, there is a

need for proper understanding of the language, even before simple program can be written and, besides, some instructions used are not really English-like.

The Fifth Generation Language (5GLs), which is also called "natural language", requires computer responding directly to spoken or written instructions or commands (Williams et al., 1999). Natural language evolved in order to bring programming much closer to human beings. Over the years, researchers have been working on designing appropriate natural languages.

Knowledge representation and translation were the initial focus of NLP with researchers exploring tasks ranging from responding to commands and answering questions in a microworld (Winograd, 1972) to responding to database queries (Woods, 1977) and understanding short stories (Schank & Abelson, 1977; Charniak & Wilks, 1976). Jelinek (1998) applied statistical method in speech recognition and this has motivated the application of similar methods to almost all areas of natural language processing. Statistical methods have now been applied in machine translation, information extraction, syntactic analysis and word disambiguation (Manning & Schutze, 1999). Some recent works were on empirical semantic role analysis (Fillmore et al., 2000; Gildea & Jurafsky, 2002). Tang & Mooney (2000) presented an approach to learn non-deterministic parsers. This area still needs to be explored in order to identify alternative approaches to learning semantic parsers.

Further investigation revealed a focus on learning to parse natural language into semantic logical forms, specifically mapping natural language sentences into programming expressions (Tang & Mooney, 2000; Tang, 2003). Some natural language projects are currently handled by European Chapter of the Association for Computational Linguistics and Microsoft Natural Language Processing Group (MNLPG). Some of the projects by MNLPG are Paraphrase, MindNet, Machine Translation and Corpus (`http://research.microsoft.com/nlp/`). Most of these projects are based on application on artificial intelligence to natural language design.

Unfortunately, there have been limited works on learning to map complete sentences to programming codes or semantic representation. However, detailed semantic analysis might be difficult for complex systems. In order to avoid this problem, it is important to think of how to build robust systems for simpler tasks, which is a major interest of this work.

3. Design of H++

As it was mentioned earlier, H++ is designed to introduce a natural language platform that is truly high in abstraction and natural. The design procedure of H++ is described here.

3.1. Significant Considerations

The goal of the work is to design and build a programming platform that will accept, analyze, understand, and translate very high-level instructions (languages that humans use naturally). The idea is to bring programming to a level of addressing computer as if we were addressing another person. Some specific considerations are:

a. to provide a programming platform that is very easy to use, even for non core programmers.
b. to provide fast access to computing functionalities without thorough coding.
c. to make programming with very few lines of code really possible. For example, writing a program to display "This is H++" 10 times in C++ gives

```
int main()
{ for int i=1, <=10, i++)
  cout << ''This is H++'';
}
```

```
Using a 4GL like Dbase, we have

I=1
K=10
Do while I<=10
      @K,10 say ''This is H++''
      K=K+1
      I=I+1
Enddo
```

The first program is 4 lines and the second program is 7 lines. One of the major interests in H++ is to solve problems like this, using just one line of instruction.

For example,

```
Display ''This is H++'' 10 times
```

d. to provide a higher level of interaction between the user and computer.

3.2. H++ Architecture

The *H++* architecture is 3-level and is shown in Figure 1 below.

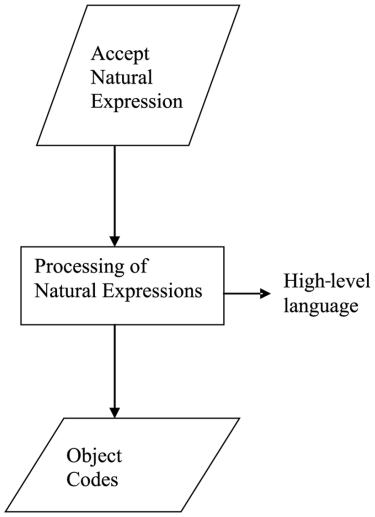

Fig. 1. H++ Architecture.

This architecture is built on a high-level language which is used to convert and compile natural expressions. Real natural languages are accepted at the first level and are translated to high-level language equivalents. Then, the base high-level language will process the translated natural expressions through the normal compilation process.

3.2.1. Program Execution Process

The program execution process (simply known as compiling process) is described in Figure 2 below.

Each program may contain one or more sentences or instruction lines, all separated with full stops (.). Each sentence is (then) stored in an array. There is a loop to process all the sentences in the whole program.

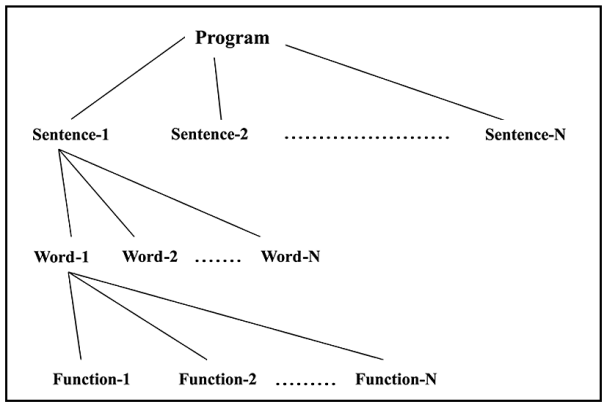

Fig. 2. Program execution process in H++.

a. Processing of Sentences

When a sentence is read into an array, it is broken down into words. In this case, a word can be either a verb (reserved word or statement) or an identifier. Words are recognized with blank spaces, which are used between words as separators. The words recognized are also stored in a different array.

There is a table (the verb table) that stores all the verbs in H++. The first word in every sentence is compared with the series of verb stored in the table. If there is a match, H++ checks

Verbs	Formats	Verb Cateogories	Operation Factors
Add	v,i,i,i,i,. v,i,i,i,.,v,i	0	+
Sub	v,i,i,i,i,. v,i,i,i,.,v,i	0	-
Display	v,i,i,. v,i,c,v	2	N
Square	v,v,i	0	^2
Square root	v,v,v,i	0	^.5
Raise	v,i,v,v,v,i	0	^.5
If	v,e,e v,e,v,e	1	C
While	v,e,e	1	C
Average	v,i,i	1	+,/

Table 1. Structure of the Format Table.

the format for that word. It simply identifies the first verb in an instruction or sentence and validates the format in the format or parsing table. Table 1 shows the structure of some of the parsing methods used in H++. The table is big enough to accommodate up to 51 verbs.

The verb column represents all the verbs in H++. The format column represents the specific format for the verbs. H++ verbs are categorized into 3 types. The third column denotes the verb category. Also, there are some operators attached to each H++ verb. Letters v,i,c,e in the format specification represent verb, identifier, constant and expression respectively. The dot (.) after i means that it can allow more identifiers.

If a given format is found, the system (then) executes the sentence by generating codes in VB. In executing a sentence, all identifiers and verbs in the front of the verb are used to generate the VB codes. There is a function for each verb in H++ that actually converts H++ sentences to VB codes.

For example,

Add A,B to C

b. H++ Function Architecture

The sentences in H++ are divided into 3 categories viz;

i. *Expression sentences*: These sentences are known as arithmetic sentences and have associated arithmetic operators. These sentences are categorized as '0' (See Table 1). Examples are the sentences that start with Add, Sub, Ave and so on.

ii. *Statement sentences*: These have no associated arithmetic sentences and are used for special and direct operations. This category is represented by "2". Examples are Display, Extract, Search and so on.

iii *Condition sentences*: These are condition sentences and are categorized as "1". Examples include While, If, and so on.

— If the sentence category (sent_cate) = 0, then the variables/identifiers within the sentence and operation factors (See Table 1) for the category are used to form arithmetic expressions. The general form of expressions in this category is Result=var1 θ var2 θ var3 $\theta \ldots \theta$ varn
Where θ represents the associated operation factor. For example, Add A,B to C would give
Result=A+B+C
C=Result
(Notes: - Result of the last arithmetic operation in H++ is always kept in a variable called RESULT.
The variables in H++ program are read as var1, var2, var3 and so on.)

— If the sentence category (sent_cate) = 1, then the condition statements are used directly because they are used in H++. So, in this case, there is no serious translation from H++ to VB.

— If the sentence category (sent_cate) = 2, then the function for the verb is called. There are different functions for verbs in this category. For example, the functions Display and Extract have functions display() and extract() respectively. These functions now generate codes based on this verb.

3.2.2. Other Design Issues

a. Variables and Identifiers

The beauty of the design is in that it allows various forms and lengths of identifiers. For example, "Victor Sodiya" can be taken to be an identifier in H++. The simple way this is done is that any combination of characters before the next verb is found and taken as identifier. However, if two or more separate names are used as a name, any of these separate names can be the verb.

Therefore, H++ will accept identifier names like

Father's Name

Is this your house?

And so on.

This enables identifiers to be formed in most natural way.

b. Rules in H++

i All H++ sentences/instructions must be terminated with a full stop e.g. Find the addition of a and b.

ii Identifiers within a sentence must be separated with "comma" and "and". For example, multiply first number and second number.

iii All H++ sentences/instructions must start with a verb which describes the action or task to be performed e.g. solve the area of a rectangle = length x breadth.
iv All H++ sentences/instructions can only consist of simple and related sentences and not compound or complex sentences. H++ can allow a sentence like add 10 and 20 and divide the result by 2. It cannot allow an instruction like add 10 and 20 and add 40 and 50.

c. Error handling

The system flags syntax errors when the rules are not obeyed or when the usage format does not exist and returns back to the instruction line.

d. Ambiguity handling

A grammar that has more than one parse tree generating a given string of tokens is said to be ambiguous (Ahoo and Ullman 1988). The architecture of H++ is simple and straightforward. As it was stated earlier, all sentences must start with a verb. Table 1 shows that each verb has a separate method of parsing or format specification and translation procedure. There are even different functions to handle different verbs. Since the key aspect of H++ sentences is the verb and there are no two verbs with the same name, then it is not possible to have ambiguity. Consequently, increasing the size of the vocabulary will only result in introducing more functions to handle the added verbs.

e. Handling of complex expressions

Complex expressions can be handled by breaking down the expressions into smaller units as it is done in the convectional programming. In H++, there could be interaction between the previous and the next expression. As mentioned earlier, the result of the previous operation is kept in a variable called RESULT. RESULT can then be used as a variable in the next expression, making it possible not to use verbose sentences to compute complex sentences.

4. H++ Interface

As it has been mentioned, H++ was implemented on Visual Basic 6.0. H++ opening menu has 8 options and several sub-options viz:- File, View, Format, Class, Method, Code, Run, Help. This is shown in Figure 2. Details of usage of H++ are available in H++ manual (Sodiya and Ojeanelo, 2005).

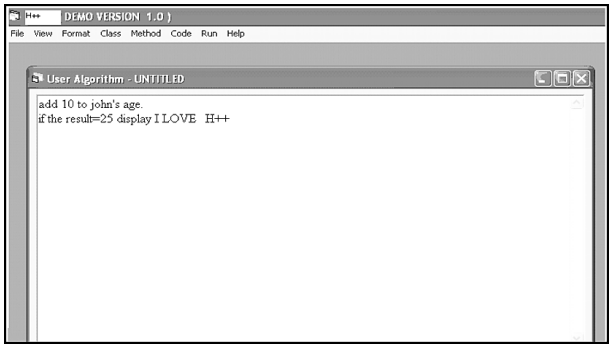

Fig. 3. H++ Interface.

5. Some H++ Statements

The usages of some H++ statements are described below:

a. Add/Sum
i. Add var1, var2 [,var3] [and var4]
ii. Add var1 to var2
In both cases, the result would be kept in a variable called result or answer. Sum can be used in place of Add

b. Subtract/Minus
i. Subtract var1, var2 [,var3] [and var4]
ii. Subtract var1 from var2
Minus can be used in place of Substract

c. Average
i. Find the average of var1, var2 [,var3] [and var4]
ii. Find the average of x numbers
Where x stands for any integer

d. Accept/Input
Accept var1, var2 [,var3] [and var4]

e. Display/print
Display var1, var2 [,var3] [and var4]

f. Solve/Find/Calculate/perform
Solve formular
where formular is an equation

g. Divide
Divide var1 by var2

f. Multiply
Multiply var1 by var2

g. Initialize
Initialize var1 = var2

h. Store result/Answer
Store result/answer as var1
Where result is the result from an arithmetic calculation

i. Square/Square root
i. Square of var1
ii. Square root of var1
iii. Find the Square/Square root of var1

j. Raise
Raise var1 to power of var2

k. Increase/Decrease
Increase var1 by var2

l. Join
Join var1, var2 [,var3] [and var4]

m. Go to/move to
Go to line number
Where line number is an integer number in front of any executable statement
The Go to and Move to are used in the same way

n. If/Should
If <condition1> <action1>
If <condition1> then <action1>
Where condition1 and action1 is an arithmetic operation
Also multiple condition and action must be enclosed in braces
Should can be used to replace If.

o. While/Until
While <condition1> <action1>

p. Extract Right/Left
i. Extract num1 from the left/right of var2
Extract a letter from the left/right of var2
Where var1 is an integer

ii. Extract blank spaces right/left
Extract blank spaces from the left/right of var1

q. Convert
Convert var1 to lower case/upper case/small letter/capital letter

r. Pause
Pause for var1 seconds/minutes/hours

s. Stop/End
End (program execution)
Stop [any word(s) to describe your action]

t. Draw
i. Draw line ab [with/using, colour type colour]
ii. Draw diagonal ab [with/using, colour type colour]
where ab represents two points in your program and colour type represents any colour such as red, pink etc.
iii. Draw triangle abc [with/using colour type colour]
iv. Draw rectangle parallel gram abacd [using/with colour type colour]

u. Move Shape/Object
i. Move shape/object abcd var1 times
Where abacd represents the coordinate enclosing the object
Var1 is an integer which represents the distance to which the object moves.
ii. Move shape/object abacd to the left/right/up/down at var1 km/hr var2 times
Where Var1 and var2 are integers

v. Separate
Separate sentence into words/letters

We have just given about one third of the total statements or verb accepted by H++.

5.1. Examples of H++ instructions

i. Solve simple interest=(principal rate of time of deposit)/100
ii. Multiply principal, rate and time of deposit.
iii. Divide the result by 100
iv. Store the result as simple interest
v. Display "I love H++" 10 times.
vi. Find the square of R
vii. Raise R to the power of 2 multiply the result by PI
viii. Perform the addition of 2 numbers twice
ix. Accept first name and last name
x. Display first name and last name convert name to capital letters

xi. Join first name and last name
xii. Extract a letter from the left of surname
xiii. Initialize level = 100
xiv. Increase result by 30
xv. While the number < 10 increase number by 1
xvi. Display the number of repetition 10 times
xvii. Pause for 10 seconds

6. Future Work

H++ can only perform operation that starts with a verb. The design can be improved further so that it will allow sentences that do not necessarily start with a verb. Though, this is an enormous task as it will definitely result in huge number of rules to handle language ambiguity. I also recommend research on mapping natural language sentences into detailed semantic and logical forms for complex systems. We hope to improve this work so as to design a natural language the world is looking forward to see.

7. Conclusion

Designing a programming language is almost the same as designing a natural language. Understanding a language means, among other things, knowing what concepts a word or sentence stands for and how to link those concepts together in a meaningful way. This goal of designing H++ was not easy to reach and the process of really putting it into shape is still on-going. We have been able to present an architecture that provides a shift from the present form of semi-English-like expressions to real English-like expressions. That is, we have introduced a more natural language – H++. This language is very easy to use and learn and it can be used by non core programmers for simple analysis and coding. H++ ideas and concepts will be useful for researchers in natural language design and processing.

References

[1] A. V. AHOO AND J. D. ULLMAN, Principles of Compiler Construction, Addison-Wesley Series In *Computer Science and Information Processing*, World Student Series Edition.

[2] B. BUNCH AND A. HELLEMANS, (2004). *History of Science and Technology*, Published by Houghton Mifflin Company.

[3] E. CHARNIAK, (1972). Towards a Model of Children's Story Comprehension, *Technical Report TR266*, Artificial Intelligence Laboratory, Massachusetts Institute of Technology.

[4] E. CHARNIAK AND Y. WILKS, (1976). *Computational Semantics*, Amsterdam: North-Holland.

[5] C. J. FILLMORE, F. C. BAKER AND H. SATO, (2000). The FrameNet Database and Software Tools, In *Proceedings of the Third International Conference on Language Resources and Evaluation* (LREC), pp. 1157–1160.

[6] D. GILDEA AND D. JURAFSKY, (2002). Automated Labeling of Semantic Roles, *Computational Linguistics* 28(3), pp. 245–288.

[7] F. JELINEK, (1998) *Statistical Methods for Speech Recognition*, Cambridge, MA: MIT Press.

[8] C. D. MANNING AND H. SCHUTZE, (1999). *Foundations of Statistical Natural Language Processing*. Cambridge, MA: MIT Press.

[9] R. C. SCHANK AND R. P. ABELSON, (1977). *Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures*, Hillsdale, NJ: Lawrence Erlbaum and Associates.

[10] A. S. SODIYA AND V. OJEANELO, (2005). *H++ Usage Manual, Version 1.0*, Vol. 1, 2005.

[11] L. R. TANG, (2003), *Integrating Top-down and Bottom-up Approaches in Inductive Logic Programming: Application in Natural Language Processing and Relational Data Mining*, Ph.D. Dissertation, Department of Computer Sciences, University of Texas, Austin, TX.

[12] L. R. TANG AND R. J. MOONEY, (2000). Automated Construction of Database Interfaces: Integrating Statistical and Relational Learning for Semantic Parsing, In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Proceeding and Very Large Corpora* (EMNLP/VLC-2000), pp. 133–141.

[13] B. K. WILLIAMS, S. C. SAWYER AND S. E. HUTCHINSON, (1999). *Using Information Technology: A Practical Introduction to Computers & Communications*, Third Edition, published by Irwin/ McGraw-Hill, pp. 507–544.

[14] T. WINOGRAD, (1972). *Understading Natural Language*, Orlando, FL: Academics Press.

[15] W. A. WOODS, (1977). Lunar Rocks in Natural English: Explorations in Natural Language Question Answering, In Zampoli, A., ed., *Linguistic Structures Processing*, New York: Elsevier North-Holland.

[16] `www.PCMECH.com`

Received: October, 2005
Revised: March, 2006
Accepted: March, 2006

Contact address:
Adesina Simon Sodiya
Computer Science Department
University of Agriculture
Abeokuta, Ogun State
Nigeria
e-mail: sinaronke@yahoo.co.uk

DR. ADESINA S. SODIYA is a lecturer in Computer Science, University of Agriculture, Abeokuta, Nigeria. His research interests are network security, artificial intelligence, natural language processing and knowledge management. He has published in both local and international journals.