A Speech Recognition based Approach for Development in C++

Mubbashir Ayub[†] and Muhammad Asjad Saleem[††]

Faculty of Telecommunication & Information Engineering
University of Engineering & Technology Taxila, Pakistan

ABSTRACT
Software development using programming languages requires keyboard for input and all programming languages are mostly text oriented. A person having brilliant mind and potential for programming skills, but suffering from RSIs or being disabled could not become a programmer. To be a good programmer a human must memorize the syntax and keywords of a programming language. In our research work we propose a methodology for C++ programming language where a programmer will speak and code in C++ will be written accordingly. Structure of special program constructs will also be created simultaneously.

Key words:
Automatic speech recognitions, Speech API, Shared Recognizer, Object Tokens, Text to Speech

1. INTRODUCTION

Programming environments can create frustrating barriers for the growing numbers of software developers that suffer from repetitive strain injuries (RSI) and related disabilities that make typing difficult or impossible. Not only is the software development process comprised of fairly text-intensive activities like program composition, editing and navigation, but the tools used for programming are also operated textually. This results in a work environment for Programmers in which long hours of keyboard typing is impossible.

Memorizing the syntax of a programming language can be frustrating for programmers because it distracts them from the abstract task of creating a correct program[1]. Visually impaired programmers have a difficult time with syntax because managing syntactic details and detecting syntactic errors are inherently visual tasks. As a result, a visually impaired programmer can spend a long time chasing down syntactic errors that a sighted programmer could have found instantly. Programmers facing repetitive stress injuries can have a difficult time entering and editing syntactically detailed programs from the keyboard. Novice programmers often struggle because they are forced to learn syntactic and general programming skills simultaneously. Even experienced programmers may be hampered by the need to learn the syntax of a new programming language.

The primary tool used for programming is a specialized text editor [2]. Early text editors were manipulated entirely via keyboard, and leveraged many keyboard shortcuts for common operations in order to speed the programming process. Unfortunately, keyboard shortcuts are usually composed of an alphanumeric key combined with one or more modifier keys (e.g. Control, Option, Alt), which contributes to RSI when keyed unergonomically. These days, text editors are usually embedded in integrated development environments (IDEs) that provide the programmer with integrated services, such as compilation and debugging [1].

One way to reduce the amount of typing while programming is to use speech recognition. Speech interfaces may help to reduce the troubles of RSI among computer programmers. At the other side speech programming may increase access for those already suffering motor impairments. Many disabled programmers are already bootstrapping voice recognition into existing programming environments [3]. However, speech does not map well onto the available applications and programming tasks. Our research uses a principled approach from field of programming languages to allow developers to use speech with much more powerful control.

We come up with a solution that addresses the above mentioned problems. Our solution requires a microphone attached to the computer. User should have all the basic understanding of C++ language. He will speak his code that will be in English language but conforms to C++ syntax and semantics. Our system will only provide code writing feature. After writing code user will copy and paste this code to any C++ compiler and will compile code in that compiler using voice commands.

2. OVERVIEW

Our system is based on Microsoft platform. System is developed using Microsoft Visual studio 2010. Windows speech recognition is used to capture voice and Microsoft Speechlib is used to convert voice to text. This converted text is then used to generate C++ code. Section 3 describes the system architecture in detail. Section 4 states the

Manuscript received October 5, 2012
Manuscript revised October 20, 2012

system comparison with other existing systems. Section 5 concludes our work with discussion of future directions.

3. SYSTEM ARCHITECTURE

Our system is divided into the five modules as shown in [Figure.1].

- Graphical User Interface (GUI)
- Voice to Text Converter
- Code Generator

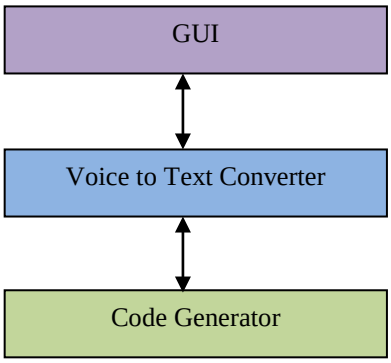

Figure.1 System Architecture

Before getting into the details of the system, let's have a brief overview of the high level working of our system. GUI module provides an interface for users to interact with the system. Voice to text converter is responsible for converting each listened word to text. A user will speak code using microphone and text converter will convert this spoken code to text. For this converted text C++ syntax and semantics are applied to this text to generate standardized C++ code.

3.1 Graphical User Interface (GUI)

GUI gives the visual appearance of the virtual file system to the end user. GUI color schemes, layout, working and behavior are quite similar to Windows Explorer. Windows XP style task pane provides easy access to common operations and gives appealing look. Standard Toolbars, popup menus and shortcut keys make operation of software easy for all type of users. Easy to Use, Easy accessibility to functions and Appealing appearance are the main features of GUI.

3.2 Voice to Text Converter

The core part of our system is voice to text conversion. We used Windows Speech Recognition(WSR), which is built on top of SAPI, for capturing voice and Microsoft Speechlib API for converting this voice to text.

SAPI version 5.4 is shipped with windows 7 and supports two distinct types of speech recognition; dictation and command and control. In our research we used dictation type of speech recognition. In this type of speech recognition machine listens to what we say and attempts to translate it into text. The accuracy of dictation ties directly to the CPU's speed and the system's available memory. The more resources, the more contexts that can be considered in a reasonable amount of time the more likely the resulting recognition will be accurate.

SAPI supports two types of recognizers: inprocess recognizer (SpInprocRecognizer) and shared process recognizer (SpSharedRecognizer). The inprocess recognizer claims resources for the application, so, for example, once an inprocess recognizer claims the system's microphone, no other application can use it. A shared recognizer runs in a separate process from the application and, as a result, it can be shared with other applications. This allows multiple applications to share system resources (like microphone). In our application we are using shared process recognizer because shared recognizer allows an application to play nicely with other speech enabled applications on system.

A recognition context is an object that manages the relationship between the recognition engine object (the recognizer) and the application. A single recognizer can be used by many contexts. For example, a speech enabled application with 3 forms will likely have a single engine instance with a separate context of each form. When one form gets the focus its context becomes active and the other two forms contexts are disabled. In this way, only the commands relevant to the one form are recognized by the engine. A single recognizer can be used by many contexts. For example, a speech enabled application with 3 forms will likely have a single engine instance with a separate context of each form. When one form gets the focus its context becomes active and the other two forms contexts are disabled. In this way, only the commands relevant to the one form are recognized by the engine. SAPI is smart enough to create the shared recognizer object for us automatically when the SpSharedRecoContext is created. In our scenario we are using dictation type of speech recognition. For this purpose we created a grammar object and load the grammar with SLOStatic value to set the dictation top of grammar as static. To set this grammar object to use dictation type of speech recognition we initialize SpeechRuleState state property of grammar object to SGDSActive.

In recognition event handler the ISpRecoResult interface is used by our application to retrieve information about the SR engine's hypotheses, recognitions, and false recognitions. The most common use of the ISpRecoResult

interface is retrieval of text recognized by the Speech Recognizer. The ISpRecoResult interface also supports the retrieval of the original audio that the SR engine recognized. An application can set interest in the SR engine's failed recognitions by calling ISpEventSource::SetInterest with SPEI_FALSE_RECOGNITION. If a false recognition occurs, the application can examine the audio (or even a partial recognition result) to reprocess the recognition or attempt to process the partially recognized text. SAPI does not require that an SR engine send a phrase with the false recognition event. ISpPhrase::GetText retrieves elements from a text phrase. All text recognized is then converted to lower case for the purpose of code generation because C++ is case sensitive and all reserved words are in lower case.

3.3 Code Generator

Code generator is the module that actually generates C++ code from listened words. As a first step, we find a list of reserved words of C++. Now for each reserved word we find words with similar sound. For example “while” have following words with similar sound. "lloyd", "while", "white", "wine" and "voice ".

After this, for each reserved word we developed a separate list structure of words with similar sound in C#. When a user speaks a reserved word that word will be converted to text and this text will be matched to the elements of list structure. If a match is found converted text is replaced with that reserved word. If no match is found text is written as it is. Now if this text is wrong and user wants to remove that word user will speak “incorrect”. A list structure is also maintained for same utterances of “incorrect”. If spoken word is matched with that same utterance then that word is removed.

At the same time if a match is found and that reserved word has special program construct then that program construct is also generated simultaneously.

Input: number sign
Output: #
Input: inglewood
Possibilities: include, in the York, in the U, in your, in lieu of, improved, etiquette moved, and you'd, intuit, inglewood, continued,in queue that, in a keynote ,in queue of, in view of the , in a view of, includes, any queue, you;
Output: include
Input: audio stream
Possibilities: iostream, all U.S. aid, audios shipping, isp and, isdn, isbn, audio stream, i a stream, on the Santiam, what a shame, i esteem, i his team, players seem;
Output : <iostream.h>
Input: void main function
Output: void main(){
}
Input: See out
Ouput: cout<<”
Input: sea in
Output: cin>>
Input: For
Ouput: for
Input: cool
Possibilities: loop, cool, who ,knew, know, move;
Ouput: for(; ;){
}
Input: Int I equals to zero;
Output: for(int i=0){
}
Input: I less than ten
Ouput: for (int i = 0; i < 10) {
}
Input: I plus plus
Ouptput: for (int i = 0; i < 10; i++) {
}

This process will continue until user completes his code. When code is complete, User will speak “select all” to select whole code. After selecting code user will speak “copy” and in C++ compiler will speak paste. All code will be pasted here. Now user will have to debug and compile this code in C++ compiler.

Figure below shows algorithm for the system.

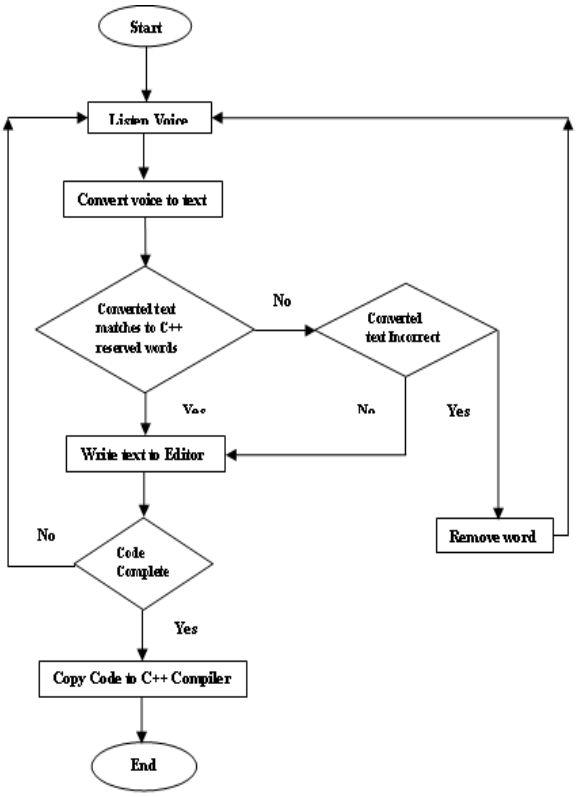

Figure.2 Algorithm

4. COMPARISON

For architectural comparison, we compared our system with various other voice programming systems available in literature e.g. VoiceGrip[8] is a programming tool designed specifically for voice input. The system allows programmers to dictate code using a pseudo code syntax that is easier to utter than the native code itself. Later, these pseudo code statements can be automatically translated in real time to native code in the appropriate programming language. For example, to type the below C statement, you will simply say:

"if one element is less than second element then"

At first, the statement would get typed literally in pseudo code, but you could later utter a single voice command to automatically translate it to the appropriate native C code.

This approach to code dictation imposes a lower vocal load on the user because it is much shorter to utter. When translating pseudo code to native code in a particular language, VoiceGrip uses a simple deterministic parsing algorithm. At each iteration, it translates a substring starting at the beginning of the remaining pseudo code. It then removes that substring from the pseudo code and proceeds with the next iteration. The process continues until there is no more pseudo code to translate. The utterance is translated in a single pass with no backtracking. This translation of each substring to native code makes translation process too much slow.

SpokenJava [6] system takes the form of a program editor called SPED and associated program analysis framework called Harmonia which are both embedded in the Eclipse IDE. A user begins by speaking some program code in Spoken Java into the editor. Once it has been processed by the voice recognizer, it is analyzed by Harmonia. Harmonia can recognize, handle and support ambiguities through the syntactic phases of program analysis as well as execute semantic analyses to disambiguate the myriad possible interpretations of the input that the first two phases create. When semantic analysis results in several legal options, our programming environment defers to the programmer to choose the appropriate interpretation. Once the interpretations have been deduced, they are translated back into Java, and written into the editor. SpokenJava is a better system but it is targeted for Java and not for C++.

[9] Consists of adding Java support to VoiceCode. This implementation consisted mainly of adding commands (loop templates, etc.) and their spoken forms to the VoiceCode program. Where possible, I kept the spoken forms for Java consistent with spoken forms in other languages. I built in extremely common commands (println, main method) and set them up to do a lot of automatic typing for the user. Two major limitations of VoiceCode are the complexity of installation and the amount of hand use involved in startup. In order to limit the amount of typing and mouse use required to start VoiceCode, I have created a batch file to start Dragon NaturallySpeaking and VoiceCode.

NaturalJava [1] is a prototype for an intelligent natural-language based user interface for creating, modifying, and examining Java programs. The interface exploits three subsystems. The Sundance natural language processing system accepts English sentences as input and uses information extraction techniques to generate case frames representing program construction and editing directives. A knowledge-based case frame interpreter, PRISM, uses a decision tree to infer program modification operations from the case frames. A Java abstract syntax tree manager, TreeFace, provides the interface that PRISM uses to build and navigate the tree representation of an evolving Java program. The goal of the NaturalJava interface is to allow programmers to write computer programs by expressing each command in natural language. But in our case user will speak native C++ code and not in natural language.

5. CONCLUSION AND FUTURE WORK

Disabled persons or Programmers suffering from repetitive strange injuries will always find it difficult for adapting to software development environments that promote long hours in front of a keyboard. Our work helps make this easier by enabling programmers to use voice recognition. In this paper we presented a robust solution for speech based programming in C++. Implementation mainly consists of finding words that have similar sound to each reserved word of C++ programming language e.g "for" have similar sound to "far", "four" and "thought". All code spoken by the user will be written in an editor. If some text is written that does not match to user intentions user will speak "incorrect" and that word will be removed. Special program constructs (e.g function structure) are also created for the sake of user and thus freeing user form the headache of remembering syntax. Our system is very suitable for disabled persons or persons suffering from repetitive strange injuries

For future development, this approach can be extended to all textual programming languages and also to visual programming languages. Similar approach can also be helpful for query writing.

Acknowledgments

We would like to acknowledge our chairman Dr. Tabassam Nawaz and our family members.

References

[1] Price, David, Riloff, Ellen, Zachary, Joseph and Harvey, Brandon. NaturalJava: A Natural Language Interface for Programming in Java. In the Proceedings of the International Conference on Intelligent User Interfaces, January 2000.

[2] Arnold, Stephen, Mark, Leo and Goldthwaite, John. Programming By Voice, Vocal Programming. In the Proceedings of the ACM 2000 Conference on Assistive Technologies. November 2000.

[3] Desilets, Alain. VoiceGrip: A Tool for Programming by Voice. International Journal of Speech Technology, 4(2): 103-116. June 2001.

[4] Yasuhrio Minami and Sadoki Farui "A large vocabulary continues speech recognition algorithm and its application to a multi-modal telephone directory system" NTT Human Interface Laboratries, Tokyo, Japan

[5] George C. Demetriou and Eric S. Atwell "Semantics in Speech Recognition and Understanding: A Survey", Artificial Intelligence Division, School of Computer Studies, University of Leeds

[6] Begel, A. Programming By Voice: A Domain-specific Application of Speech Recognition. AVIOS Speech Technology Symposium–SpeechTek West (2005).

[7] Douglas A. Reynolds "Automatic Speaker Recognition: Current Approaches and Future Trends" MIT Lincoln Laboratory, Lexington, MA USA.

[8] A. DESILETS VoiceGrip: A Tool for Programming-by-Voice INTERNATIONAL JOURNAL OF SPEECH TECHNOLOGY 4, 103–116, 2001

[9] Java Programming Using Voice Input: Christine Masuoka University of Maryland, 2008

[10] http://www.c-sharpcorner.com/Forums/Thread/75127/speech-recognition.aspx

[11] http://www.dreamincode.net/forums/topic/75990-convert-speech-to-text-and-vice-versa-in-c%23net/

[12] http://msdn.microsoft.com/en-us/library/ms720151(v=VS.85).aspx#API_Speech_Recognition

Mubbashir Ayub received his B.Sc. degree in Software Engineering from University of Engineering & Technology Taxila, Pakistan in 2007. Currently he is enrolled in M.Sc degree of Software Engineering and also serving as Lecturer in the same University in the department of Software Engineering. His areas of interests are Speech Recognition, Software Design & Architecture and Software Testing.

Engr. Raja M. Asjad Saleem did his Bachelors in Software Engineering from University of Engineering & Technology Taxila and Masters in Computer Engineering from the same in 2005 and 2009 respectively. His areas of interest are Software Specification and Design, Information Systems and Engineering Management. Engr. Raja has over 05 years teaching and research experience in University of Engineering and Technology Taxila. He has worked in different domains of technology including Computer Networks, Wireless Security. He is planning to pursue his PhD studies from a foreign university.