Java Programming Using Voice Input:

Adding Java Support to VoiceCode

University of Maryland at College Park
Department of Computer Science
Undergraduate Honors Thesis

Christine Masuoka
(cmasuoka@gmail.com)
2008
Advisor: Dr. Michelle Hugue

Table of Contents

Abstract

VoiceCode works with a commercial speech recognition program (Dragon NaturallySpeaking) and an editor (Emacs) to translate speech to code. The most recent release of VoiceCode (from Dec 22, 2006) has extensive Python and C/C++ support as well as very minimal support for Java, JavaScript, Perl, and PHP. For my honors project, I have added basic Java support to VoiceCode. 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. An installer would be useful future work, as it would make VoiceCode much easier to install.

Introduction

VoiceCode works with a commercial speech recognition program (Dragon NaturallySpeaking) and an editor (Emacs) to translate speech to code. The Dec 22, 2006 release of VoiceCode has extensive Python and C/C++ support as well as very minimal support for Java, JavaScript, Perl, and PHP (VoiceCode wiki 1).

Why programming by voice?

Speech recognition software has helped many people who have difficulty typing. However, commercial speech recognition programs are intended for the dictation of text in a natural language, such as English. Programming languages were never meant to be spoken, and most have a lot of punctuation, unusual spelling and capitalization ("println", "computeLength"), and non-standard symbols ("++"). As a result, standard speech recognition software cannot be easily used to write programs.

For example, to write a simple Java loop

```
while(i<5){
      System.out.println(i)
}
```

using commercial speech recognition, such as Dragon NaturallySpeaking, the user would have to say:

> "tab-key while uncap that open-paren i spell that choose 6 less than select less than (type '<') five close-paren open-brace new-line tab-key tab-key system cap-that dot out dot println spell that (manually delete 'line' and type in 'ln' or say "P R

I N T L N") open-paren(manually delete space before the '(') i spell that choose 6 close-paren semi-colon new-line tab-key close-brace"

Training the speech recognition software would improve the situation somewhat. For example, typing "println" could be probably be avoided if this had been previously added to the vocabulary of Dragon. Saying 'dot' avoids the addition of extra spaces and avoids capitalization of the following word, but saying "period" causes the program to add to extra spaces and to capitalize the first letter of the next word. Presumably, if the user dictated enough code and added enough class and method names (like "println") to the vocabulary and trained Dragon by reading those into the microphone, the user would not need to explicitly format those symbols, but even then, the user would still need to dictate lots of punctuation, and in the meantime the user would have to explicitly format for capitalization, correct spelling, and manually delete the spaces that Dragon automatically puts between each word.

Another problem with using Dragon "as-is" is that Dragon has difficulty with mathematical formulas. Since many written documents spell out numbers rather than using digits, Dragon is biased in favor of spelling numbers out. Dragon has very limited support for math symbols of interest for programming (+-/%, but no '<' in the manual for Dragon), even though there is a "smiley/frowny/winky face" command.

The authors of VoiceCode recognized the limitations of traditional speech recognition software and created VoiceCode to overcome these difficulties. Programming by voice allows people who cannot use their hands (both programmers with repetitive stress injuries and people with other disabilities) to program. According to an article in a Canadian newspaper, The Daily (2003), approximately 10% of Canadian adults (2.3 million people) have a repetitive strain injury. Of these 23% have RSI in the hand or wrist. Programming by voice could also be useful in preventing typing related injuries.

Why Java support?

More extensive Java support has the potential to greatly benefit programmers who have difficulty using their hands. Java is the language used for the Computer Science AP exam, and is a fairly common language for introductory computer science classes. According to the website "Programming Language Popularity", which bases its measurements on searches, open source projects, job postings on craigslist, and discussion sites, Java is one of the most popular programming languages.

Initially, VoiceCode supported only the Python programming language. For a master's degree project at the University of California, Santa Cruz, Stuart Norton added support for C and C++ in VoiceCode (Norton 2004).

For my project, I added Java support to VoiceCode by designing the spoken forms and corresponding actions then writing the necessary code in Python. I also created a small batch script to minimize the amount of typing needed to launch Dragon NaturallySpeaking and VoiceCode.

Background

VoiceCode is a project begun by the National Research Council of Canada (Désilets, et al. 2006). It works with a commercial speech recognition program (Dragon NaturallySpeaking) and an editor (Emacs) and allows programmers to produce code by speaking fairly English-like pseudocode into a microphone. The December 2006 release of VoiceCode has extensive Python and C/C++ support as well as very minimal support for Java, JavaScript, Perl, and PHP. The project is open-source and available for download from both the VoiceCode website and SourceForge (see Appendix 1: Online Resources).

A few other speech to code programs have been created. None of these are commercially available, and many are incomplete (just a prototype), not available to the public, and/or extremely limited.

In "Programming By Voice: A Domain-specific Application of Speech Recognition", Begel describes "Spoken Java, a syntactically similar, yet semantically identical variant of Java that is easier to speak. We built an Eclipse IDE plugin called SPEED (for SPEech EDitor) to support the combination of Spoken Java and an associated command language." (Begel 2005). I was unable to find the plugin.

NaturalJava only supports a subset of Java. It seems like there has been no work done on it since 2000. It is also not linked to speech recognition yet; the paper describes a Wizard of Oz prototype (Price, et al. 2000).

In "An Investigation Into Programming by Voice and Development of a Toolkit for Writing Voice-Controlled Applications", Snell discusses developing an editor (CodeTalker) to work on any platform and with any speech recognition program. The CodeTalker interface (not just code dictation) is controlled by voice. The primary goal of the project was a voice enabled programming editor, but it "provides a toolkit for developing all kinds of voice controlled software applications, not just ones to support programming." CodeTalker supports a bit of HTML and Java. There is more support for HTML than Java. There was just enough Java support to run user tests of the editor prototypes. Users can say "add replacement" to add custom commands or phrases. One of the best things about CodeTalker is that it's an all-in-one package, the only other thing you need to install is a speech recognition program. However, users need to compile their code separately in the command prompt (Snell 2000). While CodeTalker looks very promising, I could find neither source code nor a downloadable and usable finished product.

VoiceGrip is mainly a collection of macros that non-interactively change text into code. It supports the languages C/C++ and Perl and the editors Emacs and MS DevStudio. The user says "translate file/region/line", and VoiceGrip translates from English-like syntax to one of the supported programming languages, using the file extension to determine which

(VoiceGrip 3 documentation). VoiceGrip's functionality has become part of VoiceCode, which is more interactive and more powerful (VoiceCode wiki 2).

Methods and Implementation

The National Research Council of Canada posts advice on adding new language support to VoiceCode:
1.Write a define_language statement (a regular expression listing the characters that can be used in the identifiers (names of a class, variable, method, etc.) in a particular language).
2. Update editor specific code for language identification: Add the '.java' file extension is added to VoiceCode's map of file extensions so that Emacs can recognize Java files.
3. Create new templates for loops, methods, etc. Add reserved words.
4. Add symbols from standard libraries.
5. Create new symbol abbreviations. This involves identifying symbols that are not already in VoiceCode's vocabulary and creating abbreviations for them (VoiceCode wiki 3).

What had already been done

When I began this project, I found that a small amount of preliminary work on Java had already been done. Many types of Java statements were already supported by the code for "c-style languages" (C/C++, Java, JavaScript, Perl, and PHP). Another developer added a small amount of Java-specific support (define-language statement, editor-specific code for language identification, and a template for Java import statements). There was also some code for function definition in Java, which had not been linked to the main code file. None of the commands for Java were enabled.

What I implemented/changed

I used Java Precisely (Sestoft 2002) as a syntax reference and for information on formatting conventions in Java. Java 5.0 Tiger (McLaughlin and Flanagan 2004) was used as a reference on the newer (version 1.5 and later) features of Java.

I added commands either to a section of VoiceCode filled with Java commands (if they were unique to Java or had been specified in separate sections for the other languages (python C/C++)) or to the sections with general/all-languages commands (if there were similar commands already implemented for python and C/C++). Examples of commands that I added to the general/all-languages section are modulo (%), binary and/or/xor/not, if, return, and break. Examples of commands I specified separately are Java reserved words (new, super, throw, etc.), switch/case/default, and try/catch/finally. See Appendix 2 for a full list of the commands I added.

There was a minor error in the define-language statement (I fixed this). I also found a more serious bug: the code for "c-style languages" differentiates between define vs. declare method/function. While the difference is meaningful in C/C++, it is not in Java or any of the other "c-style languages".

I also added some automatic symbol formatting. To follow the generally accepted formatting conventions for Java, I set the general formatting preferences for Java to "standardLowerIntercaps" and the formatting preferences for Java class names to "StandardIntercaps".

In order to limit the amount of typing and clicking required to start VoiceCode, I have created a batch file to start Dragon NaturallySpeaking and VoiceCode (see Appendix 3). I also describe a way to get both the script and Emacs to launch when the user logs in. Users may have difficulty finding someone who will start VoiceCode for them every time they want to use it. So, this method of startup has great potential to help current users of VoiceCode, many of whom have difficulty typing.

Results

Design decisions

How spoken forms were chosen

Where possible, I kept the same spoken forms used for the other languages supported by VoiceCode. This enables users to move between languages more easily. For symbols unique to Java, I looked at how the statement or symbol was described in Java reference books. I also considered how I would pronounce it and how I had heard other people pronounce it.

VoiceCode has to balance flexibility against brevity of spoken forms. Ideally, spoken forms are unique, short, easy to figure out and remember, consistent with user expectations/standard practice within the computer science field, and consistent across languages. Unique spoken forms are best because Dragon is easily confused by homophones (for example, "4" and the "for" in a for loop).

Shorter spoken forms may help avoid voice strain.

> "A study by Kambeyand, Singer and Cronk suggests that people with a history of Repetitive Strain Injury (RSI) are more likely to suffer Voice Strain from using SR systems than people with no previous history of RSI. Also, programming being the intense activity that it is, it may lead to more Voice Strain than dictation for word processing." (VoiceCode wiki 4)

VoiceCode also supports multiple ways to dictate the same code. For example, 'do', 'do the following', 'loop body', 'body', and 'for body' or 'while body' can all be used to move to the body of a loop. This flexibility allows users to choose the spoken form that seems most natural to them.

One question is how much to type automatically and how much to have users dictate explicitly. I have "main" and "main method" produce

```
public static void main(String[] args){
^
}
```

rather than requiring the user to dictate "public static void define method main with arguments string bracket close-bracket args [go to] body". Also, "print line" produces

```
System.out.println(^)
```

However, some users may want to disable commands like these so that those spoken forms can be used for something else. Therefore, in order to make these commands easier to turn off, I added them to a separate command set from the others.

What was not supported and why

I chose not to support the 'goto' keyword because it is not supported by many recent implementations of Java. Also, using 'goto' is considered bad programming practice.

I decided not to support Java escape sequences(like \t) or Java output formatting (like %n). These are easy to dictate literally "backslash t" and "percent n". Although support for these would mean the user would not need to memorize as much, most people who know Java will have already learned the more common ones. Also, the spoken forms are likely to confuse Dragon. For example, saying "escape tab" (to get "\t") is likely to produce "`escape  `" (Dragon types the word "escape" then presses the tab-key) or "`escapeTab`" (VoiceCode interprets the words as a new variable name).

Due to time constraints, I concentrated on getting the core of the Java support to work and did not add much support for the newer (version 1.5 or later) features of Java.

Examples of Java dictation

To illustrate how the Java portion of VoiceCode works, some examples of dictation with VoiceCode are given below. In the following examples, these conventions are used:
^ indicates the cursor position.
"What the user says" is in double quotes.
[Optional parts of spoken forms] are in brackets.
`Text produce by VoiceCode is in this font.`

Example 1:
"[define] class my guitar extends basic guitar cap-that implements playable cap-that body"

```
class MyGuitar extends BasicGuitar implements Playable {
^
}
```

(Currently, unless `BasicGuitar` and `Playable` have been processed with the "compile symbols" command, they will not be capitalized, and must be explicitly formatted with the cap-that command.)

Example 2:
"double define method m with arguments integer I body return I semi" or
" double define method m with arguments integer I body new statement return I "

```
double m(int i){
      return i;^
}
```

(Note that in addition to mostly automatic punctuation, VoiceCode also indents for you. The second way of saying it will result in an extra blank line before the return statement. The spoken form "define method" could be changed to "method". However, I decided to stay consistent with the spoken forms in the other languages supported by VoiceCode, even though this results in the user saying a word, "define", that doesn't show up in the actual code.)

Example 3:
"while i less than five body new statement print line i"

```
while(i<5){
      System.out.println(i);
}
```

Discussion

Several features of VoiceCode (in both previous versions and my project) make programming by voice easier. Where possible, I used spoken forms that were consistent with those used in the other languages supported by VoiceCode, so that users can move between languages more easily. VoiceCode automatically puts in most punctuation, allowing users to program without worrying about the exact syntax. The Java support that I added formats symbols (names of variables, classes, methods, etc.) according to the formatting conventions of Java. VoiceCode also knows how to spell and format symbols that have been analyzed using the 'compile symbols' command. In addition, it automatically recognizes the link between abbreviations and unabbreviated spoken forms. For example, if you have a variable named `maxNum`, you can say "maximum number" or "max num" (or some combination of these), and VoiceCode would type `maxNum`.

During this project, I developed a method used to start VoiceCode with minimal typing and clicking (see Appendix 3). Not counting the clicks and keystrokes needed to log in, this method uses 2 clicks and 13 keystrokes. Without the script and the setup described, 3 clicks, 3 double-clicks, and 74 keystrokes are needed. The calculations count holding two keys down at the same time as two keystrokes and assume that the user has set all .java files to open in Emacs and that he or she has shortcuts to Dragon NaturallySpeaking (without the VoiceCode user specified in the shortcut) and to the command prompt.

The Java API is huge, and I did not add all of the symbols from it, both because of time limitations for this project and because the library symbols added to VoiceCode are compiled at startup (having too many could slow the loading of VoiceCode). Luckily, Java has fairly few oddly spelled symbols, and the capitalization is very regular (class names are in StandardIntercaps, method names are in standard LowerIntercaps, etc.). The Java support doe not do as much as it could to automatically format new symbols (such as superclass names and interface names). Also, I did not add support for many of the newer (version 1.5 and later) commands (generics, annotations, etc.).

For future development, in addition to adding more Java support, there are several features unrelated to Java whose addition to the program would greatly benefit the users of VoiceCode and allow more people to take advantage of the program. Support for additional editors and more extensive support for some of the languages would help work towards the project's goal of supporting all major programming languages and editors. An installer would be useful future work, as it would make VoiceCode much easier to install and would enable more people to begin using VoiceCode.

Conclusions

For my honors project, I have designed and added basic Java support to VoiceCode. Implementation consisted mainly of adding commands and their spoken forms to the VoiceCode program. Where possible, I kept the spoken forms for Java consistent with spoken forms in other languages. This will enable users to move between languages without having to learn different spoken commands for Java. I included multiple spoken forms for some of the commands (for example, `int` can be pronounced as either "int" or "integer"). I built in extremely common commands (println, main method) but put them in a separate section so that users who want to can turn them off. Java support in VoiceCode will allow more programmers to exploit VoiceCode's programming by voice capabilities and will assist disabled programmers and give non-disabled programmers an alternative to typing code.

Acknowledgements

I would like to thank my project advisor Dr. Hugue, Alain Désilets and Quintijn Hoogenboom for answering my questions by email and allowing me to contribute to the project, and Mike Quinn for installing VoiceCode for me.

References

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

Désilets, A., Fox, D.C., and S. Norton. Voice Code: An Innovative Speech Interface for Programming-by-Voice. Extended Abstracts of the 2006 Conference on Human Factors in Computing Systems (Montreal, Quebec, Canada, 2006).

McLaughlin, Brett and David Flanagan. Java 5.0 Tiger: A Developer's Notebook. Sebastopol, CA: O'Reilly, 2004.

Norton, Stuart. Master's Degree Project: Contributing to the Voice Code Programming-by-Voice Toolbox. http://voicecode.iit.nrc.ca/VCodeWiki/uploads/Adding_new_language_support_to_the_VoiceCode_Programming_by_Voice_Toolbox.pdf (2004).

Price D. et al. NaturalJava : A Natural Language interface for Programming in Java. In Proc. of the Int. conf. on Intelligent User Interface (2000).

Programming Language Popularity. http://www.langpop.com/ (accessed June 27, 2008).

"Repetitive Strain Injury." The Daily. 12 August 2003. http://www.statcan.ca/Daily/English/030812/d030812b.htm (accessed July 10, 2008).

Sestoft, Peter. Java Precisely. Cambridge, MA: The MIT Press, 2002.

Snell, L. An Investigation Into Programming by Voice and Development of a Toolkit for Writing Voice-Controlled Applications. M Eng. Report. Imperial College of Science, Technology and Medicine, London (2000).

VoiceCode wiki (accessed 2008):

1. http://voicecode.iit.nrc.ca/VoiceCode/public/ywiki.cgi (2006).
2. http://voicecode.iit.nrc.ca/VoiceCode/public/wiki.cgi?obj=VoiceGrip (2004).
3. http://voicecode.iit.nrc.ca/VCodeWiki/public/wiki.cgi?AddingSupport_for_a_NewLanguage (2007).
4. http://voicecode.iit.nrc.ca/VoiceCode/public/wiki.cgi?What_is_voice_strain (2004).

VoiceGrip 3 documentation: http://voicecode.iit.nrc.ca/VoiceCode/uploads/VG3/ (1998, accessed 2008).

Appendix 1: Online Resources

The source code for VoiceCode is distributed under the GNU General Public License and can be viewed and downloaded from:
http://sourceforge.net/cvs/?group_id=4120

The VoiceCode wiki is located at:
http://voicecode.iit.nrc.ca/VoiceCode/public/ywiki.cgi

Recent stable releases of VoiceCode are available at:
http://voicecode.iit.nrc.ca/VoiceCode/public/ywiki.cgi?Downloading_and_installingVoiceCode
These include everything, except for Dragon NaturallySpeaking, that you need to install and run VoiceCode (python, Emacs, NatLink, etc.).

A video demonstration of the stable version ofVoiceCode can be seen at:
http://voicecode.iit.nrc.ca/VoiceCode/uploads/VoiceCode_CHI_Demo_Movie.mov

Appendix 2: Java Commands Added During This Project

modulo (%)
binary and, or, not, xor (& | ~ ^)
begin comment (//)
return
break
class definition
interface definition
subclass of (extends)
class body (move cursor to class body)
try
catch
finally
switch
case
default
cast
cast to
of type
returning
System.out.println
main method (public static void main(String[] args){})

Java reserved words:

new
super
throw
volatile
void
true
transient
throws
this
synchronized
strictfp
static
short
public
protected
private
package
null
native
long

int
instanceof
implements
float
final
false
double
const
class
char
byte
boolean
assert
abstract
enum (from Java 1.5)

Note: The 'import' command was already supported.

Appendix 3: Launching VoiceCode at Startup *

On my Windows XP computer, I arranged for VoiceCode to launch at login with minimal typing and clicking using the following setup and steps.

To set up Voice Code to launch more easily:

1. Make and move these 2 things to C:\Documents and Settings\yourusername\Start Menu\Programs\Startup :

(1) shortcut to a blank .java file (Start.java)

- Make the Java file.
  - Note: If it's Python or C/C++ you're interested in working in, substitute a file with the appropriate file extension. If you're working on a specific file, you could use that in the Startup folder instead of the blank file.
- Right-click and select "Create Shortcut".
  - Note: You need to use a shortcut to the file rather than the file itself because Emacs produces a Start.java~, which will cause an error message next time you log in (because the computer doesn't know how to open .java~ files).

(2) a batch file (StartDragonVoiceCode.cmd) with the following contents:

```
cd C:\Program Files\Nuance\NaturallySpeaking9\Program\
start natspeak.exe /User VoiceCode
cd %VCODE_HOME%\Mediator
python wxMediator.py
```

- To make the batch file, type or paste these lines into Notepad. Then save as type "All files" with the file name ending in .cmd.
- Note: In the second line of the batch file, "VoiceCode" can be replaced with whatever you've named your Voice Code user in Dragon.

2. Associate all .java files to start with Emacs:
   Right-click on Start.java.
   Select Open With>Choose Program...
   Select Emacs.
   Check the "Always use the selected program to open this kind of file" checkbox.
   Click OK.

To launch Voice Code:

1. Log in.
2. Click Dragon's "microphone on" button to turn on the microphone.
3. In Emacs, Press Escape+X. Type vcode-mode and hit the Enter key.

Final note: I would recommend creating a separate account for this, unless you want to start Voice Code every time you log in. Alternately, just put the script on your desktop; so, you can launch Dragon and Voice Code easily whenever you want to, instead of at login.

* Corresponding wiki page:
http://voicecode.iit.nrc.ca/VCode_1_Doc/public/wiki.cgi?Launching_Voice_Code_at_Startup