Spoken Language Support for Software Development

Andrew Begel
Computer Science Division, EECS
University of California, Berkeley
Berkeley, CA 94720-1776
abegel@cs.berkeley.edu

Abstract

Software development environments have changed little since their origins as low-level text editors. Programmers with repetitive strain injuries and other motor disabilities can find these environments difficult or impossible to use due to their emphasis on typing. Our research adapts voice recognition to the software development process, both to mitigate this difficulty and to provide insight into natural forms of high-level interaction. Our contribution is to use program analysis to interpret speech as code, thereby enabling the creation of a program editor that supports voice-based programming. We have created Spoken Java, a variant of Java which is easier to verbalize than its traditional typewritten form, and an associated spoken command language to manipulate code. We are conducting user studies to understand the cognitive effects of spoken programming, as well as to inform the design of the language and editor.

1 Introduction

Software development environments can create frustrating barriers for the growing numbers of developers that suffer from repetitive strain injuries and related disabilities that make typing difficult or impossible. Our research helps to lower these barriers by enabling developers to reduce their dependence on typing using voice recognition (VR). Speech interfaces may help to reduce the onset of RSI among computer users, and at the same time increase access for those already having motor disabilities. In addition, they may provide insight into better forms of high-level interaction.

The initial challenges to support voice-based programming are two-fold – enabling input that is natural to speak, yet understandable by the system, and making it possible for VR tools to process it well. The main artifact of our work will be realized as a software development system that can understand spoken program dictation, composition, navigation and browsing, and editing.

2 Programming By Voice

Programming languages have historically been communicated in written form; verbalization of code is highly ad-hoc and is not used in any formal way. Through experimentation, we have found that for the most part, there does exist a vernacular for speaking programs. We also found some uses of prosody to indicate punctuation, and observed that programmers tended to identify patterns and describe them, rather than using only their instantiations. Other experiments we have conducted regarding existing VR command languages show that searching and navigating by voice requires too much input, is too slow, and incurs more cognitive load than using the keyboard or mouse [3]. This implies that a complete solution to programming-by-voice must include efficient navigation and editing mechanisms.

Guided by our results, and by studies on the use of speech, such as Shneiderman's and Karats' work [6, 10], we have designed a first version of Spoken Java. Spoken Java is a syntactic variant of standard Java language which is more naturally verbalized by human developers. It is semantically identical to Java; the only difference is in its appearance on the screen and its manner of entry.

The novel form of input in this project presents technical challenges for language design. Spoken input contains many lexical ambiguities, such as homophones, misrecognized, unpronounceable, and concatenated words. When the input is natural language, it can be disambiguated by a hidden Markov model provided by the VR vendor. However, when the input is a computer program, natural language disambiguation rules do not apply. Not only do these ambiguities affect the voice-based programmer's ability to introduce code, they also affect the ability of the programmer to use similar sounding words in different contexts. We have created new technology to address these problems.

A user begins by speaking some code into the editor. Once it has been processed by the VR, it is analyzed by our Harmonia analysis framework [4]. Harmonia can identify ambiguous lexemes in spoken input [2], and incorpo-

Proceedings of the 2004 IEEE Symposium on Visual Languages and Human Centric Computing (VLHCC'04)
0-7803-8696-5/04 $ 20.00 IEEE

rate these lexical ambiguities into a set of ambiguous parse trees which contain all structurally legal interpretations of the input. A disambiguation phase is then used to filter out the semantically illegal programs. When semantic disambiguation results in multiple legal options, our environment defers to the user to choose the appropriate interpretation.

The speech editor is being embedded in the Eclipse IDE. The programmer will edit fragments of Java code by first having them translated into Spoken Java, editing them by voice, retranslating the results back into Java, and reinserting the code into the program. In addition to composing and editing code, the programmer may perform high-level program manipulations. For example, he may invoke a program refactoring or search for a particular structural or semantic entity in the code base by saying "Find all references to the MyList dot getElement method and replace them with StandardList dot getElementAt". To support this combination of commands and code, we developed Blender, a lexer and parser generator that can merge descriptions of formally specified languages into a tool that seamlessly recognizes the combination.

We will give the user on-screen assistance to alleviate the difficulty in knowing what speech vocabulary and grammar are available. Since the user will be able to create and select transformations and other higher-level mechanisms, knowledge of language syntax does not suffice to know the vocabulary. Our interface will make visible the spoken phrases used to generate code as that code is reviewed, thereby reinforcing the available vocabulary. When the user needs to specify a location on the screen, we will use a novel context-sensitive mouse grid to numerically identify locations in the program where a particular action or statement is valid.

Evaluation will take the form of user studies of programmers composing and editing code. We will iterate the design of the interface, command language, Spoken Java, and the development environment's capabilities in response to these studies. We consider our work to be beneficial if our metrics show that users can perform better on our system than the current state-of-the-art.

3 Related Work

Efforts to apply speech-to-text conversion for programming tasks such as authoring, navigation, and modification using conventional natural language processing tools have had limited success. The tools provided by two commercial products, IBM's ViaVoice and Scansoft's Naturally Speaking, are only designed to work for natural languages. Merely speech-enabling text editors, as has been done by IBM, Scansoft, Jeff Gray at U. of Alabama (SpeechClipse) [9], and by contributors to public domain software [8], is not enough to support software development tasks. Some researchers have attempted to adapt VR for programming [5], but their work suffers from awkward, over-stylized code entry, and the inability to exploit the structure and semantics of the program. A few researchers [1, 7] have applied programming language analysis technology to understand the program code being spoken, but all have their limitations. More recent work [11] has shown that keyword-triggered code template expansion can ease some of this awkwardness. We are using this technique in our work.

4 Conclusion

Software developers with motor impairments will always have a difficult time adapting to software tools that emphasize the keyboard. Our work helps mitigate this problem by enabling programmers to utilize VR in the development process. Our user studies will help inform the design of Spoken Java and its associated spoken command language. Applying traditional compiler program analyses to spoken input enables us to create a program editor that is better suited than the state of the art to enable motor-impaired software engineers to compete effectively in the workforce.

References

[1] S. C. Arnold, L. Mark, and J. Goldthwaite. Programming by Voice, VocalProgramming. In *ASSETS*, pages 149–155. ACM, 2000.

[2] A. Begel and S. L. Graham. Language analysis and tools for ambiguous input streams. In *LDTA*, 2004.

[3] A. Begel and Z. Kariv. *SpeedNav: Document Navigation by Voice*. http://www.cs.berkeley.edu/ abegel/speednav9.pdf.

[4] M. Boshernitsan. Harmonia: A flexible framework for constructing interactive language-based programming tools. Technical Report UCB/CSD-01-1149, EECS – U. of California, Berkeley, 2001.

[5] A. Desilets. *VoiceGrip 3*. http://ai.iit.nrc.ca/il_public/VoiceCode.

[6] C. A. Halverson *et al.* The beauty of errors: Patterns of error correction in desktop speech systems. In *Proceedings of INTERACT'99*, Speech, pages 133–140, 1999.

[7] D. Price *et al.* NaturalJava: A natural language interface for programming in Java. In *Proceedings of IUI*, Short Paper/Poster/Demonstration, pages 207–211, 2000.

[8] T. V. Raman. Emacspeak – direct speech access. In *ASSETS*, pages 32–36, 1996.

[9] S. Shaik *et al.* SpeechClipse: an Eclipse speech plug-in. In *Proceedings of the 2003 OOPSLA workshop on eclipse technology eXchange*, pages 84–88. ACM Press, 2003.

[10] B. Shneiderman. The limits of speech recognition. *Communications of the ACM*, 43(9):63–65, Sept. 2000.

[11] L. Snell. 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, June 2000.