Idiolect: A Reconfigurable Voice Coding Assistant

Breandan Considine
McGill University
bre@ndan.co

Nicholas Albion
Independent Developer
nalbion@yahoo.com

Xujie Si
University of Toronto
xsi@cs.utoronto.ca

Jin Guo
McGill University
jguo@cs.mcgill.ca

***Abstract*—This paper presents Idiolect, both an IDE plugin for voice coding and a novel approach to building bots that allows for users to define custom commands on-the-fly. Unlike traditional chatbots, Idiolect does not pretend to be an omnicient virtual assistant but rather a reconfigurable voice programming system that empowers users to create their own commands and actions dynamically, without rebuilding or restarting the application. We present a case study of integrating Idiolect [1] with the IntelliJ Platform, illustrate some example use cases, and offer some lessons learned during the tool's development.**

***Index Terms*—speech recognition, voice programming, bots**

I. Introduction

Humans are able to learn new words and phrases, and apply them in a variety of contexts relatively quickly. This is currently not the case for chatbots, which are often limited to a set of static commands and phrases defined at compile-time. This is a burden to bot-developers as well, who must anticipate user intent and write bindings for each new use case. On both sides, this presents a time-consuming and expensive challenge, resulting in an impedence mismatch between author expectations and user intent.

On the other end of the spectrum are general-purpose scripting and metaprogramming languages that allow users to define their own commands and build embedded domain specific languages (eDSLs). These systems are highly flexible, but require a high upfront investment from the user, who must design a language to define their own commands, and then learn the language itself, all whilst doing their daily job as a software engineer.

However, there is a fecund middle ground between these two design choices, that allows users to quickly dictate their own commands and phrases without resorting to a Turing Complete language. For example, the user might say "whenever I say *open sesame*, open the settings menu", and the system will learn this command and open the settings menu whenever the user says "open sesame". Or "whenever I say *redo thrice*, repeat the last action three times". Or call a function in a scripting language, open a file, or anything else the user can think of.

This flexibility addresses a common problem in the field of voice UX design, where users are unable to express their intent in a way that the system understands. For example, a user may want to open a specific file, but the system only understands the command "open file". The user must then learn the system's command vocabulary, and then rephrase their intent in a way the system understands. This experience can be a frustrating one, and often results in users abandoning the system altogether.

Idiolect defines a default lexicon of phrases, but does not force users to learn them. Instead, we allow users to define their own commands and phrases on-the-fly, and then incorporate them into the system immediately. This shifts the burden of learning from the user to the system, which can then learn the user's idiolect and adapt to their specific needs, freeing users to express their intent in a way that is most natural to them.

In this paper, we describe Idiolect, a dynamically reconfigurable system that allows users to create their own commands and actions on the fly, by verbally expressing the desired behavior of the system. This imposes some natural constraints, because the user must be able to express their intent in a way that can be verbally communicated. By targeting IDEs, whose users are already familiar with programming, any action that requires complex instructions can be written programmatically, and then invoked on the fly with a keyword or phrase.

Primarily, Idiolect observes the following design principles: (1) be natural to use, (2) be easy to configure, (3) get out of the user's way as quickly as possible. We believe that these principles are important for a system that is intended for developer-use, who are busy people and more than capable of configuring the system themselves. We also support motor-impaired users who have difficulty typing, or prefer to use a voice interface.

II. Prior Work

Prior work in this area has explored...

III. Action Binding

The IntelliJ Platform has over 10^3 possible actions. These actions are bound to keyboard shortcuts, menu items, and toolbar buttons. The user can also bind actions to voice commands. However, the user must first know the name of the action, and then bind it to a voice command. The default grammar was manually curated from this list, using the name to generate a description that is suitable for voice recognition.

IV. Command Prioritization

Highest priority commands are those that enable and disable speech recognition.

Then, user-defined commands.

Then the default commands from a plugin-wide grammar.

[1] https://github.com/OpenASR/idiolect

The recognizer of last resort are ChatGPT commands. We can use a prompt "What action is the most likely for the phrase "..." out of these actions: ..." and it will select top action as the command.

V. Speech Models

Speech recognition, the problem of translating an audio waveform containing speech to text, is essentially a solved problem. We can use one of the many existing speech recognition models, and they will work well enough for our purposes. However we also want a free and open source real-time offline speech recognition platform, which has only recently become available on commodity hardware.

Idiolect integrates with Vosk, a state-of-the-art deep speech system with models for various languages. VoskAPI[2] is open source system that can be used a Java library, which we use. For TTS, we use the built-in voices from the parent operating system, via the jAdapterForNativeTTS[3] library.

VI. Intent Recognition

Idiolect supports a variety of methods for acting on a user's utterance, which can be defined by string matching, a context-free grammar, and LLM prompting. This framework forms an extensible DSL for the creation of custom patterns to match against transcribed speech, which can be defined by an end-user via a simple configuration file, or programmatically by a plugin developer to handle more complex usage scenarios.

The plugin first attempts to perform an exact lexical match, by attempting to resolve a given utterance against a predefined lexicon. This is the primary way to define and modify commands, and is typically most reliable way to match a command previously known to user. However, the user may not know or recall the phrase to which an action they wish to perform was bound.

In this case, next attempts to match the speech against a grammar, which can be a regular expression that is matched against the utterance, and that may contain named capture groups, used to extract parameters from the utterance. For example, we can match more complex patterns, such as "open the (?¡file¿.*) file in the (?¡project¿.*) project". This is a powerful tool for developers, who can define their own grammars to match against user utterances, and extract parameters from the utterance.

Finally, if the utterance does not match any of the predefined lexicons or grammars, the plugin will attempt to match the utterance against a language model (LM). This is a probabilistic model of the language, which can be used to predict the most likely utterance given a sequence of words. For example, the utterance "I want to edit foo.java" is more likely to be the command "open foo.java" than the command "execute foo.java".

[2]https://github.com/alphacep/vosk-api

[3]https://github.com/jonelo/jAdapterForNativeTTS

VII. Error Recovery

Idiolect supports defining and recognizing context free grammars. In keeping with the design principles of "parse, don't validate", we allow users to define their own grammars, and then bind them to actions. However there are many domain-specific terms that can be difficult to parse directly.

As a first line-of-defense, we incorporate Considine et al.'s (2022) work on Tidyparse. In short, if a given phrase "open foo.java" is uttered, but the word file is missing, we repair the string to "open file foo.java". This is done using a Levenshtein automata, which attempts to find the smallest edit transforming a string outside the language to a string inside the langauge. The algorithm is a form of error recovery, where we attempt to locate the most likely parse tree that approximately matches the user's utterance.

We also support a form of error-recovery, where Vosk returns us a list of alternate utterances, and we attempt to match each of them against the lexicon and grammar. This is a form of error recovery, where we attempt to locate the most likely parse tree that approximately matches the user's utterance, given the list of alternate utterances.

Finally, we can use a language model to rerank the most likely utterances, conditioned on a previous context of historical commands. This is a form of error recovery, where we attempt to locate the most intent that approximately matches the user's utterance, given the dictionary, context and a list of alternate utterances.

We describe each of these in more detail in the following sections.

VIII. Large Langauge Models

IX. Plugin Extensions

X. Conclusion

References

[1] G. Eason, B. Noble, and I. N. Sneddon, "On certain integrals of Lipschitz-Hankel type involving products of Bessel functions," Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955.

[2] J. Clerk Maxwell, A Treatise on Electricity and Magnetism, 3rd ed., vol. 2. Oxford: Clarendon, 1892, pp.68–73.

[3] I. S. Jacobs and C. P. Bean, "Fine particles, thin films and exchange anisotropy," in Magnetism, vol. III, G. T. Rado and H. Suhl, Eds. New York: Academic, 1963, pp. 271–350.

[4] K. Elissa, "Title of paper if known," unpublished.

[5] R. Nicole, "Title of paper with only first word capitalized," J. Name Stand. Abbrev., in press.

[6] Y. Yorozu, M. Hirano, K. Oka, and Y. Tagawa, "Electron spectroscopy studies on magneto-optical media and plastic substrate interface," IEEE Transl. J. Magn. Japan, vol. 2, pp. 740–741, August 1987 [Digests 9th Annual Conf. Magnetics Japan, p. 301, 1982].