Idiolect: A Reconfigurable Voice Coding Assistant

Breandan Considine [1] Nicholas Albion [2] Xujie Si [3]

Abstract

Idiolect[1] is an open source [2] tool for voice coding and a novel approach to building chatbots that lets users define custom commands and grammars on-the-fly. Unlike traditional chatbots, Idiolect does not impersonate an obedient assistant, but instead offers a highly-configurable interface for automating repetitive programming tasks. Featuring advanced support for error correction and audio-visual feedback, Idiolect promises to enhance the usability and accessibility of manual developer tools by offering an alternate input method for keyboardless programming. The following paper explores the mechanics of voice programming in Idiolect, and sheds light into the challenges faced and insights gained during its development.

1. Introduction

Humans are able to quickly learn new words and phrases, and apply them in a variety of contexts. Traditional chatbots, however, are often limited to a set of static commands. This creates a frustrating experience for users, who struggle to express their intent, as well as bot developers, who must anticipate user intent and make new capabilities discoverable. This rigidity is a common source of misalignment between user intent, author expectations and bot capabilities.

One direction to address this problem is to improve intent recognition, however, in this paper, we explore the design and implementation of a reconfigurable chatbot for programmers. Idiolect is a programmable voice assistant designed for developers working in an integrated development environment (IDE) or a general-purpose programming context, with a focus on usability, configurability and accessibility.

Idiolect provides a default vocabulary of phrases, but does not impose them upon end-users. Instead, users may configure custom voice commands on-the-fly, which are incorporated into the vocabulary instantly. By shifting the burden of adaptation from the user to the system, this frees our users to express their intent in a more natural manner.

[1] https://youtu.be/R4TDx8GAa-c

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

Primarily, Idiolect observes the following design principles: be (1) natural to use, (2) easy to configure, (3) as unobtrusive as possible. We believe that these principles are important for a system intended to be used by developers, who are busy people and quite capable of configuring the system themselves. We also support developers with visual and motor impairments, who may have difficulty typing, or prefer to use a voice interface for the sake of convenience.

2. Background

Early attempts to build keyboardless programming systems can be traced back at least two decades to Leopold & Ambler (1997), later revisited by Arnold et al. (2000), Begel & Graham (2005) and others. These systems allow users to write code by speaking into a microphone, however early voice programming systems were limited by a small vocabularies. Another stream of work from Chkroun & Azaria (2019) targets teachable voice user interfaces, but unlike our work, do not consider IDE integration or configuration. It also predates many of the recent breakthroughs on large language modeling, which we consider a transformative enabling technology for the adoption of voice programming.

Although related, voice assistants such as Siri and Alexa are not configurable nor intended for programming. Our approach is more akin to a programmable bot with audio I/O than a dedicated voice assistant, which are often more distracting than helpful when programming. Spoken programs are also closely related to natural language programming, an idea once scorned (Dijkstra, 1979), but now becoming increasingly plausible, thanks to the emergence of large language models. Shaw (2022) calls for new programming languages to address the needs of "vernacular developers".

3. Architecture

Today, automatic speech recognition (ASR), the translation of an audio waveform containing speech to text, is essentially a solved problem – one of the many pretrained ASR models would work well enough for our purposes. By default, Idiolect integrates with Vosk, a state-of-the-art deep speech system with realtime models for offline recognition. This allows us to provide a high-quality speech recognition experience without requiring users to share personal data.

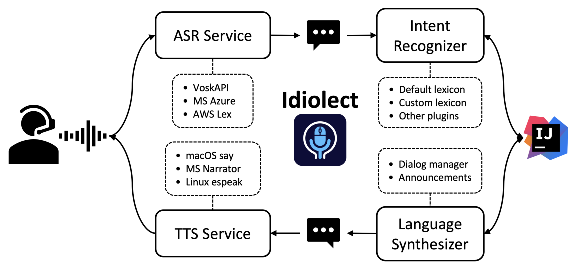

Figure 1. Idiolect has four major components: a speech and intent recognizer, language synthesizer, and TTS service.

Once a spoken utterance is decoded as text, Idiolect must determine the relevant actions and entities needed to fulfill the command. Furthermore, it may need to consider the IDE context, i.e., the current editor state and command history, to resolve potentially ambiguous commands. For example, the phrase, "open plugin menu" could refer to multiple different menus, depending on when and how it was invoked.

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 a voice command directly to an action, presuming the user already knows the action's identifier. Idiolect's default grammar was manually curated from the IDE action list, using the CamelCase identifier to generate a suitable description for intent recognition.

Idiolect dispatches utterances to a series of recognizers using a turn-based priority queue, in which each recognizer is given a single turn to match or pass on each utterance. Once a command is recognized, the command is consumed and dispatched to no subsequent recognizer thereafter, which prevents a single command from triggering multiple actions.

4. Barriers and Pathways to Usability

We have observed some common usability challenges arise during the development of Idiolect and will now discuss some of those obstacles and our efforts to overcome them.

4.1. User Onboarding

Several users reported confusion when first installing our plugin. To address this issue, we added a wizard that guides users through installation. Upon first installing the plugin, a user is greeted and prompted to download the Vosk model for recognizing their preferred natural language, configure the properties file, and bind a few voice commands.

4.2. Plugin Observability

Recognition failure is a common issue, often caused by transcription errors in the ASR model, due to, e.g., noise in the audio signal, stopwords, or poor recognition accuracy. To address this issue, we added a visual cue that displays the phrase transcribed, any action triggered upon intent recognition, and the raw audio waveform. This allows users to more easily diagnose and correct recognition errors.

4.3. API Extensibility

Idiolect tries to anticipate users actions and expose default action bindings, however certain functionality is best left implemented by downstream developers, who are better-equipped to understand their users' needs. In addition to end-user configurability, Idiolect is designed to be extensible by other IntelliJ Platform plugins, for defining recognition handlers and custom actions. We offer a simple message-passing API for plugins to communicate with Idiolect, and a DSL for defining custom commands and grammars.

4.4. Error Recovery

Another common usability barrier is when transcription is accurate, but the utterance does not correspond to an actionable command, possibly due to stopwords, verbal fillers or extraneous text which cannot be directly parsed. To correct recognition errors, we apply Considine et al. (2022), which supports recognition and parsing of context-free and mildly context-sensitive grammars, and computing language edit distance. We use a SAT solver to find the smallest edit transforming a string outside the language to a string inside the language. Only when the utterance is one or two tokens away from a known command do we attempt a repair.

VoskAPI is also capable of returning a list of alternate utterances alongside a confidence score for each, which can be used to determine if any recognized utterance is sufficiently close to an actionable command in the Idiolect grammar.

5. Conclusion

On one end of the design spectrum are embedded domain specific languages (eDSLs). These systems can be powerful, but require a high upfront investment from the user. On the other end are static languages, which can be challenging to maintain and extend. In the middle of this language design spectrum are *idiolects*, which give users the freedom to create, share and reconfigure conversational domain-specific languages for their own idiomatic needs. By targeting developers, who are typically comfortable with configuration languages, commands with more complex semantics can be expressed programmatically, then invoked on-the-fly.

We believe the intersection between conversational agents, vernacular programming languages and programmable voice assistants to be fertile ground, and one relatively unexplored in the language design space. Hopefully, our small contribution inspires others to explore and take similar steps towards making programming more natural and accessible.

References

Arnold, S. C., Mark, L., and Goldthwaite, J. Programming by voice, VocalProgramming. In *Proceedings of the fourth international ACM conference on Assistive technologies*, pp. 149–155, 2000.

Begel, A. and Graham, S. L. Spoken programs. In *2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC'05)*, pp. 99–106. IEEE, 2005.

Chkroun, M. and Azaria, A. Lia: A virtual assistant that can be taught new commands by speech. *International Journal of Human–Computer Interaction*, 35(17):1596–1607, 2019.

Considine, B., Guo, J., and Si, X. Tidyparse: Real-Time Context-Free Error Correction. In *Workshop on Live Programming*, 2022. URL https://github.com/tidyparse/tidyparse.

Dijkstra, E. W. On the foolishness of natural language programming. *Program Construction, International Summer School*, pp. 51–53, 1979.

Leopold, J. L. and Ambler, A. L. Keyboardless visual programming using voice, handwriting, and gesture. In *Proceedings. 1997 IEEE Symposium on Visual Languages (Cat. No. 97TB100180)*, pp. 28–35. IEEE, 1997.

Shaw, M. Myths and mythconceptions: What does it mean to be a programming language, anyhow? *Proceedings of the ACM on Programming Languages*, 4(HOPL):1–44, 2022.