Beach Wreck Ignition: Challenges in open source voice

If you're anything like many open source enthusiasts, you may have grown up watching science fiction shows like Knight Rider, or Star Trek, or my personal favorite – Time Trax. What do all of these have in common? In each of them, voice is the key medium through which the protagonists interact with a computer. Knight Rider had Kitt, Star Trek had the ubiquitous computer, and even the indefatigible Darrian Lambert in Time Trax had dependable Selma, the holographic assistant.

As advances in machine learning, compute power and neural networks have been made, voice interaction is increasingly moving from science fiction into science fact.

Disappointingly however, the early leaders in this space have primarily been commercial and proprietary players - Apple, with Siri, Amazon with Alexa, Microsoft with Cortana, Google with Home, Samsung with Bixy. Given these for-profit imperatives, it is no surprise that voice data is now routinely recorded and stored by big companies, then used to segment audiences, profile people and target them with advertising. Voice data, like other big data, is now a currency.

Given the widespread adoption of natural language understanding as a user interface, and the possible privacy intrusions from proprietary solutions, traction is starting to grow for open source voice.

So, what does an open source voice stack consist of?

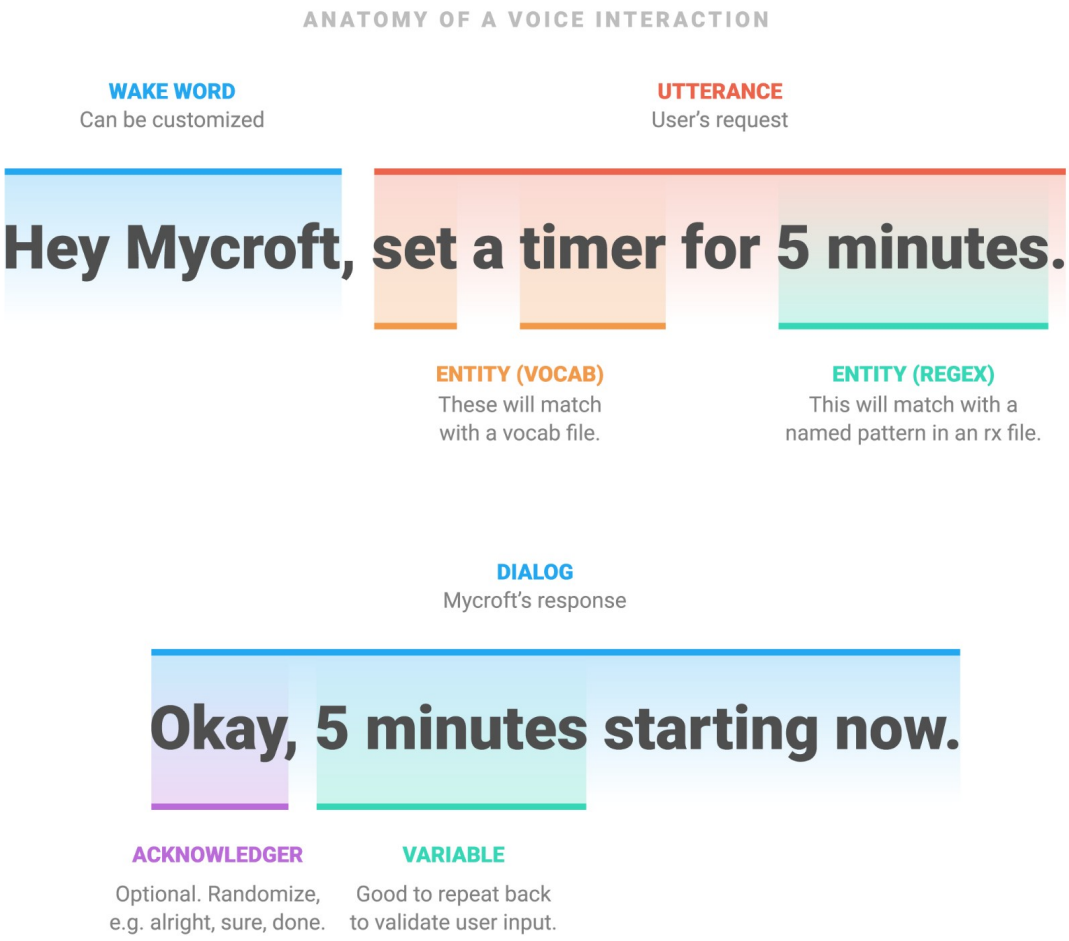

Image credit: Mycroft AI used with permission

In a voice interaction you begin with a Wake Word - also called a Hot Word - which is used to prepare the voice assistant to receive a command. Next, we use a Speech to Text engine to transcribe an Utterance from voice sounds into written language. Next, we use an Intent parser to determine what _type_ of command the user wanted to execute. Then, we select a command to run and execute it. Next, we turn written language back into voice sounds again using a Text to Speech engine.

At each layer of the voice stack, there are several open source solutions available.

Wake Word detection

One of the earliest Wake Word engines used was PocketSphinx. PocketSphinx recognises Wake Words based on something called *phonemes*. A phoneme is the smallest unit of sound that distinguishes one word from another in a particular language. Different languages have different phonemes – for example, it's hard to approximate the Indonesian trill "r" in English (this phoneme is the sound you make when you "roll" your r sound). Using phonemes for Wake Word detection can also therefore be a challenge for people who are not native speakers of a language – as their pronunciation may differ from the "standard" pronunciation of a Wake Word.

Snowboy is another Hot Word detection engine – available under both commercial and open source licenses. Snowboy differs from PocketSphinx in that it doesn't use phonemes for Wake Word detection; it instead uses a neural network that is trained on both false and true examples to differentiate between what is and isn't a Wake Word.

Mycroft AI's Precise Wake Word engine works in a similar way – by training a recurrent neural network to differentiate between what is and isn't a Wake Word.

All of these Wake Word detectors work 'on device' - meaning that they don't need to send data to the cloud, which helps to protect privacy.

Speech to Text

Accurate Speech to Text conversion is one of the most challenging parts of the open source voice stack.

Kaldi is one of the most popular Speech to Text engines available, and it has several "models" to choose from. In the world of Speech to Text, a "model" is a neural network that has been trained on specific data sets, using a specific algorithm. Kaldi has models for English, Chinese and some other languages too. One of Kaldi's most attractive features is that it works "on-device".

Mozilla's DeepSpeech implementation – part of their broader Common Voice project – aims to also support a wider range of minority languages. As at the time of writing, the compute requirements for DeepSpeech mean that it can only be used as a cloud implementation – it is too "heavy" to run 'on device'.

One of the biggest challenges with Speech to Text is training the model. Mycroft AI have partnered with Mozilla, enabling our community to help train DeepSpeech. We then pass the trained data back to Mozilla to help improve the accuracy of their models.

Intent Parsing

A strong voice stack also needs to ensure that the intent of the user is accurately captured. There are several open source intent parsers available.

Rasa is open source and widely used in both voice assistants and chatbots.

Mycroft AI uses two intent parsers. The first, Adapt, uses a keyword matching approach to determine a confidence score, then passes control to the Skill, or command, with the highest confidence. Padatious takes a different approach, where examples of *entities* are provided, so that it can learn to recognise an *entity* within an *utterance*.

One of the challenges with Intent Parsers is that of intent collisions – imagine the utterance;

“Play something by The Whitlams”

Depending on what commands or Skills are available, there may be more than one that can handle the intent. How does the Intent Parser determine which one to pass to? At Mycroft AI we have recently implement our Common Play Framework, which assigns different weights to different entities, leading to a more accurate overall intent confidence score.

Text to Speech

At the other end of the voice interaction lifecycle is Text to Speech. Again, there are several opensource TTS options available. In general, a TTS model is trained by gathering recordings of language speakers, using a structured corpus – or set of phrases. Machine learning techniques are then applied to synthesize the recordings into a general TTS model – usually for a specific language.

MaryTTS is one of the most popular, and supports several European languages.

Espeak has TTS models available for over 20 languages, although the quality of synthesis varies considerably between languages.

Mycroft AI’s Mimic TTS engine is based on CMU Flite and has two voices available for English. The newer Mimic 2 TTS engine is a Tacotron-based implementation, which is a less robotic, more natural sounding voice. Mimic runs on-device, while Mimic 2, due to compute requirements, currently runs in the cloud. Additionally, Mycroft AI have recently released the Mimic Recording Studio, an open source, Docker-based application that allows people to take recordings which can then be trained using Mimic 2 into an individual voice.

This solves one of the many problems with TTS – having natural-sounding voices available in a range of genders and languages / dialects.

Parting notes

As you can see, there is an emerging range of open source voice tools becoming available, each with their own benefits and drawbacks. One thing is for certain though – the impetus towards more mature open source voice solutions that protect privacy is here to stay!

But several challenges remain.

Kathy will be talking about challenges in open source voice at linux.conf.au on Thursday, 24th January 2019 - https://linux.conf.au/schedule/presentation/148/