# A Novel Kiribati Language Learning Tool

David Morley

April 16, 2020

## Abstract

A proof of concept of a Kiribati language learning tool is developed. The tool is comprised of a dialogue engine for practicing greetings and basic conversations with an associated text to speech engine to read the responses back to the user. This rudimentary tool mimics speaking with a native when a native speaker is not available for practice. The fully integrated tool is not completed, but the constituent parts are completed to a satisfactory level. Methods of developing the software and possible areas for future improvement are detailed.

## 1 Introduction

### 1.1 Project Description

Learning how to communicate in a new language is a challenging problem that requires much patience on the part of the language learner and teacher. It also requires much practice with fluent speakers. The complexity increases when working with a small language with few resources and even fewer native speakers available to teach and help practice. This is the situation that prospective learners of the Kiribati language face. This project intended to create a tool to provide an outlet for Kiribati language learners to practice their conversation without a fluent speaker present. In order to simulate a native speaker, there are two main capabilities that such a tool needs to have. It needs to know what words/phrases to use in conversation as well as know how to correctly pronounce those words. As an initial step to answer this need, I propose the creation of a novel Kiribati dialogue move engine (DME) capable of greetings and basic conversations with an associated text to speech (TTS) engine so that the machine can say its responses to the user. With such a tool, novice speakers could practice simple dialogue and hear the generated responses without a native speaker present.

### 1.2 Motivation

A tool of this nature could be very useful for anyone wishing to learn to communicate in Kiribati. This would be true of prospective missionaries wanting learning resources before they enter the Missionary Training Center or as a resource to augment learning while at the MTC. It could be similarly useful for returned missionaries wanting to bring family or friends back with them. Similarly, it would be helpful to those wanting to practice their language skills while in a quarantine situation.

Further, the work required to build this application would enable much learning in the areas of speech synthesis and dialogue processing; both in terms of technology and linguistic analysis.

The demand for speech applications is growing rapidly throughout the world. Any former experience with these technologies is extremely marketable in job markets. Additionally, due to its nature as a relatively small and largely undocumented language, much of the linguistic work for defining the language has never been done before. Any advancement in the linguistic analysis of this language, as well as most tools in this language, are completely groundbreaking.

### 1.3 Related Work

The technologies at play in this tool are not new. There are several open-source tools available for both of the main tasks. The most common tools for creating speech synthesizers are MaryTTS and Festival. Common for building dialogue move engines are OpenDial and Unreal Engine. Additionally, as demand for speech applications grows, more language bases are wanting access to these technologies. As a result, much research has been done on building text to speech generators for small languages; see for example "TTS From Zero: Building Synthetic Voices for New Languages" a thesis by John Kominek.[1] In the course of researching these technologies and other work, I have not been able to find any record of either of these technologies being built for this specific language. Thus, even though preexisting platforms and research were leveraged, all work done was completely original and novel.

## 2 Resources

### 2.1 Systems

As previously mentioned, this tool consists of two main parts; namely a dialogue move engine and a speech synthesizer. Thus, my approach consisted of two major efforts. I used Festival as the framework system for the TTS engine. Festival was primarily written by Alan W. Black, Paul Taylor and Richard Caley from the University of Edinburgh and Carnegie Mellon University.[2] Lengthy documentation and tutorials have been provided to facilitate using this software. However, many holes in the documentation exist and individual runtime errors took a significant amount of the development time of this tool. Further, the linguistic data and sound recordings needed to be developed then fitted to the provided framework. Festival version 2.4 was used for this project.

OpenDial was chosen as the framework to build the DME on as it was designed to be paired with custom TTS engines. This tool was originally developed by the Language Technology Group of the University of Oslo (Norway), with Pierre Lison as the main developer.[3] It is a Java-based tool that uses dialogue domains published in XML. OpenDial version 1.5 was used in this project.

### 2.2 Data

In this proof of concept version of the tool, all that is needed for the DME is the conversation tree. A simple tree was developed to allow the user to have brief conversations in a

[1] https://www.lti.cs.cmu.edu/sites/default/files/research/thesis/2009/john_kominek_tts_from_zero_building_synthetic_voices_for_new_languages.pdf
[2] http://festvox.org/docs/manual-2.4.0/festival_3.html#Acknowledgements
[3] http://www.opendial-toolkit.net/

limited domain. The domain focused on common greetings and basic conversation topics. This tree was stored in a .xml file for use by OpenDial.

According to the Festival documentation, three things needed are needed to define a voice for a TTS engine: a diphone database, a lexicon (and its rules), and several scheme files for the voice[4]. Compared to English, Kiribati is a relatively small language with few exceptions to rules. Given its alphabet of 13 characters, this language has a reasonable number of monophones. By my calculations, there are about 72 basic monophones. The vast majority of diphones are formed by the direct concatenation of these monophones. This provides a feasible set of phones to physically record and thereby build the requisite database. Festival provides a framework for defining diphones by describing rules for allowable letter pairs. Using Festival, these rules were used to create a list of nonsense words containing two diphones each. See **Figure 1** for an example of these nonsense words. Only 71 such nonsense words were needed to provide a sufficient basis to synthesize this language. All recordings were provided by myself at no cost.

```
( gil_0026 "pau t a t a t a pau"    ("t-a"  "a-t") )
( gil_0027 "pau t a k a k a pau"    ("k-a"  "a-k") )
( gil_0028 "pau t a b a b a pau"    ("b-a"  "a-b") )
( gil_0029 "pau t a m a m a pau"    ("m-a"  "a-m") )
( gil_0030 "pau t a n a n a pau"    ("n-a"  "a-n") )
( gil_0031 "pau t a ng a ng a pau" ("ng-a" "a-ng"))
```

Figure 1. A sample of the nonsense words used for recording diphones. The first column shows the filename where the recording was stored, the second shows the word that was recorded, and the third displays the phones contained in the sample. In this case, “pau” denotes silence.

The formation of the lexicon consisted of two main steps: defining the possible sounds and associating those sounds with specific letter combinations. These sounds are referred to as phones and were defined by the categories and definitions found in **Table 1.**

| Category | Possible Definitions |
|---|---|
| vowel or consonant | yes, no, or irrelevant |
| vowel length | short, long, diphthong, or schwa |
| vowel height | high, middle, or low |
| vowel frontness | high, middle, or low |
| lip rounding | yes, no, or irrelevant |
| consonant type | stop, fricative, affricative, nasal, liquid, or approximant |
| place of articulation | labial, alveolar, palatal, labio-dental, dental, velar, or glottal |
| consonant voicing | yes, no, or irrelevant |

Table 1. A list of the phone categories and corresponding definitions

[4] http://festvox.org/festvox-1.2/festvox_15.html

With the phones defined, individual sounds could be defined. Since Kiribati is a phonetically consistent language, only a few word-specific entries were made in the lexicon. Most words were defined through letter to sound rules. Punctuation was defined to be silent and numerals were not addressed.

The third component of the text to speech system, as defined by Festival, is the scheme files for the voice. These files provide information for the indexing of the sound files and playing them back in a reasonable manner. All files were generated automatically from the diphone recordings. Some; however, had to be manually adjusted as the computer was not accurate. Without any outside aid, the software:

- extracted pitch marks
- extracted Linear Predictive Coding (LPC) coefficients
- added prosody information about phrasing and intonation

Festival also automatically computed phone durations and the phone boundaries from the recordings. Unfortunately, at least a third of these were completely erroneous and needed to by manually and painstakingly updated.

# 3 Methods

As it was assumed to be the more time-intensive task, the TTS engine was developed first. Only once it was sufficiently functional did work commence on the DME. This section will provide a brief narrative of the methods implemented regardless of how successful they were. This is done in hopes that certain shortcomings may be avoided in future work.

## 3.1 The Windows Failure

The first and several successive attempts to install Festival all ended in failure. This was due to the large incompatibilities between Festival and any Windows system. The operating system in use was Windows 10. The underlying framework of Festival is written in C++. This means that although the developers tried to make the software operating system agnostic, they were largely unsuccessful because of the everchanging versions of C++ produced by the various vendors of the language.[5]

It is possible to build Festival on a Windows system, but much care needs to be taken in obtaining compatible versions of ported Unix based tools. A tutorial on by Cameron Wong on eGuideDog enabled me to get a sufficiently working copy of Festival on my machine.[6] With this version, I was able to code in the linguistic features and make the sound recordings. The recordings were done in one of the high-quality sound recording booths in the Harold B. Lee Library at Brigham Young University. Two recordings of each nonsense word were made, but only the clearer and more correctly pronounced version of each diphone was kept. This was to ensure that

[5] http://festvox.org/festival/index.html
[6] https://www.eguidedog.net/doc/doc_build_win_festival.php

the diphones were all of a high quality even though the person making the recordings was not a native speaker.

Even with the success in recording and preparing the system, nothing could be built without a working installation of the Edinburgh Speech Tools Library (Speech Tools). After emailing him, Cameron Wong was kind enough to provide me with a copy of Speech Tools, but alas, it did not port well. Thus, after several hours of attempting to make it work, I broke down and installed an Ubuntu 18 Linux virtual host on my machine.

## 3.2 Linux Success

I was able to get several weeks' worth of progress on my Windows machine done in a matter of hours on my Linux machine. As is expected, there were still many bugs encountered due to the challenges of working with a code set developed by someone else. However, the caliber and frequency of these bugs were markedly diminished after switching operating systems. With a working version of Speech Tools installed, the scheme files were generated relatively quickly.

After months of work, the system finally spoke its first words on April 11, 2020. It was quickly realized; however, that the system was drawing from the wrong sounds. Festival uses a concatenative approach to speech synthesis. When the system is instructed to say a sequence of words, it uses highly optimized algorithms to quickly break down the words into letter pairs and retrieve the associated scheme information. Among other things, it reads the power information and intonation files to normalize sound output. It also accesses the index file detailing the timestamps of the phones in the recordings. At least a third of these automatically generated timestamps were pointing to completely different phones in the recordings. Many others needed to be fine-tuned to make the synthesized speech understandable.

Further, there was a misunderstanding when the feasible letter pairs were defined. It was assumed that the only letter pairs needed were those that could appear inside a word. However, to make the robot voice flow better, Festival also uses recordings of diphones across word boundaries. These recordings, as well as recordings containing silence for the beginning and ends of sentences needed to be created. Some new recordings were made using a consumer-quality webcam, but most new information was squeezed (sometimes incorrectly) out of the existing recordings.

These circumstances resulted in a voice synthesizer that produces understandable speech but that is not perfect. For example, there is a loud crackling every time the letter 'i' is used. Also, some phones sound especially warbled which is a result of stretching the data to include more sounds than it actually did. With more recordings and more fine-tuning, this system should go from merely understandable to being not uncomfortable to listen to.

There are also still some noticeable bugs or errors in how the language was defined in the Festival framework. For this reason, any word with an 'ng' letter pair is deemed "unpronounceable" by the system. Further, the speech synthesizer does not offer support for capitalized letters. Thus, capitalized letters at any place in a word deems the entire word "unpronounceable" as well. Workarounds for these errors have been developed by adding common

‘ng’ words directly to the lexicon and requiring all inputs to be in lower case. However, future versions will require more sustainable fixes to these and other issues that may arise.

### 3.3 The Dialogue Move Engine

Contrary to the process of building a new voice in the Festival system, constructing a dialogue system is an intrinsically less defined problem. The original intent was to make a system that would guide the user through a narrow conversation tree. The depth, breadth, and content of that tree were completely open-ended and allowed me a lot of room for creativity. OpenDial was chosen as the foundation for the DME as it was designed to be paired with custom TTS engines.

Unfortunately, OpenDial cannot prompt users with possible responses. This feature would be necessary to guide new speakers through possible conversation trees and to teach them correct language principles. As a result, the system cannot support a strong pedagogical framework and needs to be relatively robust in responding to a variety of inputs. Perhaps future iterations of this tool will use a different tool to account for the first problem. OpenDial accounts for the second issue by using a Bayesian framework for response selection based on the clarity of the input.

I designed a naïve dialogue tree for use in OpenDial that only can perform the most rudimentary of conversations. This was due in part to the scope of this project being to merely provide a proof of concept model of the idea as well as a result of the need to complete a final product in the allotted time. **Figure 2** contains a possible conversation that could be held with this current system. As this system is designed to teach new language learners, the domain is centered on new conversation behavior. It responds to a variety of greetings, health and activity inquiries, and farewells.

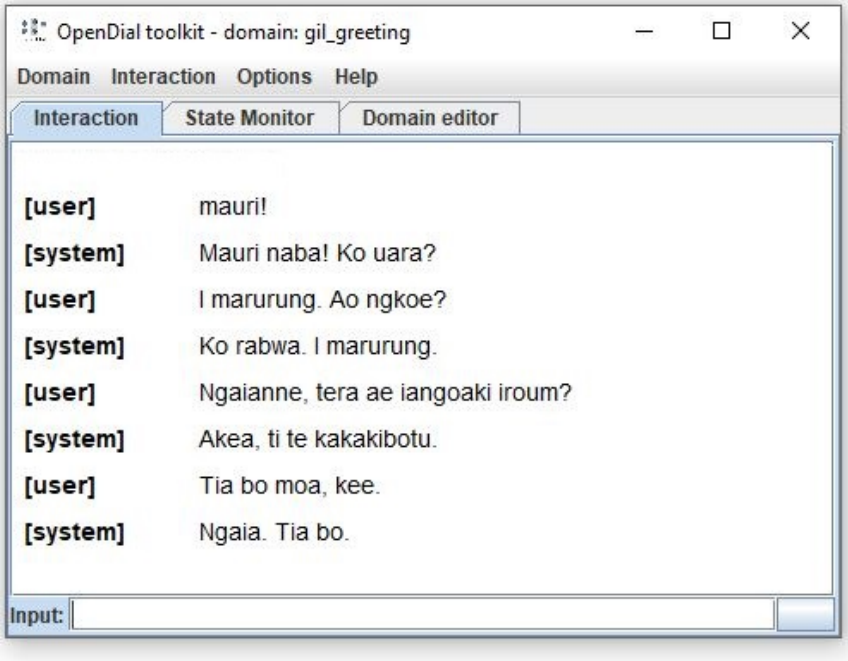


Figure 2. A demonstration of the conversational range of the programmed agent.

The system is sufficiently robust to respond to a variety of different ways of asking these same questions. That said, any diversion from these shown topics above will receive a confused response from the system, see **Figure 3.**

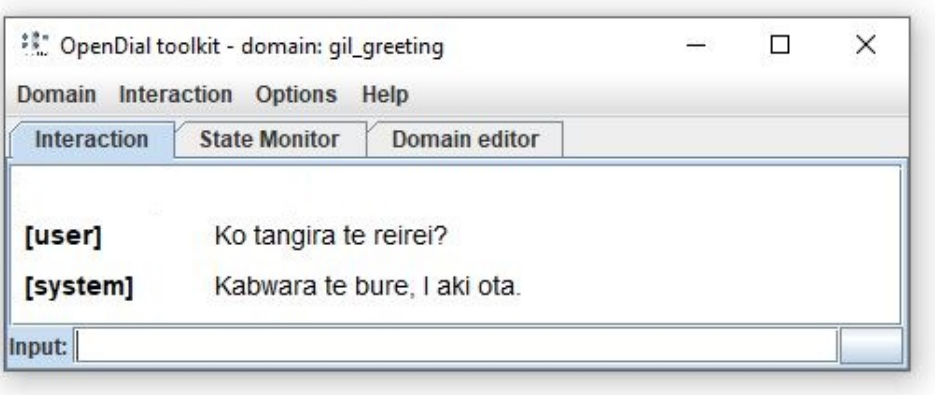


Figure 3. Any question not regarding the aforementioned topics is responded to without understanding.

### 3.4 Incorporating Speech Synthesis in the DME

The final step of the project as presented was to put the pieces together. The fully functioning tool needs to have audible responses to statements to simulate a real conversation environment. OpenDial has support for a speech synthesizer built with MaryTTS, another open-source tool for working with text to speech engines. The Festival voice created in section 3.2 would need to be archived in such a way as to be ported into MaryTTS. However, getting MaryTTS to automatically read the OpenDial responses turned out to be more complicated than initially expected. Work on this phase was suspended to a combination of poor documentation, package version compatibilities, and deadlines.

## 4 Future Work

Other areas of work have already been alluded to. The final integration of the OpenDial conversational agent and the Festival voice remains to be completed. Further, the dialogue tree needs to have the ability to talk about more topics and give more varied responses before it will be a truly useful tool. Work could be done to create a Kiribati language speech recognition system to make the agent have more naturally delivered conversations. Also, more work should be done on the synthesizer developed to give it more capability and natural sound. Lastly, once these improvements are made, the tool could be distributed to the target audience of potential Kiribati language learners.

## 5 Conclusion

While it is nigh impossible to become truly fluent from a single resource, some level of prowess can be attained. The created tool is meant to represent a minimum viable product that demonstrates paths for future tools as well as to provide useful functionality. As they currently stand, both the TTS engine and the DME outlined are overly simplistic. However, a tool to guide users through this certain type of introductory conversation is very useful for new language

speakers. This type of conversation is held with almost every person encountered. Lastly, this project gave me exposure to new areas of natural language processing I have not had before. The things I have learned this semester are directly applicable to my interests in developing smart assistants and voice technology. Having worked with speech recognition last semester, I now have a thorough introduction to the generation and conversational aspects of automated agents enabling me to get involved in future projects that will be even more interesting to me.