Voicebuilding seminar 2016 report

group #2: Fraser Bowen, Yauhen Klimovich, Nataniel Borges Jr

This report describes steps we made to build English voice using MaryTTS.

Slides for seminar can be found here. This report can be used as a guidline to build your own voice.

For those who just want to get it Fraser's voice up&running, do the following:

1. `git clone https://github.com/YauhenMinsk/voicebuilding_seminar_16.git`
2. run `./gradlew legacyInit` (~3-6 min on a modern laptop 8Gb RAM) OR download voice in tarball to the root of the cloned folder and extract tarball (it should be build/ folder extracted). If you got BUILD SUCCESS, do the next item.
3. run the `./gradlew run` (~10-13 min on the same env). Wait until you see something like the following

```
016-05-04 10:33:58,054 [main] INFO  marytts.main Startup complete.
 started in 3.432 s on port 59125
2016-05-04 10:33:58,399 [main] INFO  marytts.server Starting server.
2016-05-04 10:33:58,460 [main] INFO  marytts.server Waiting for client to connect
on port 59125
```

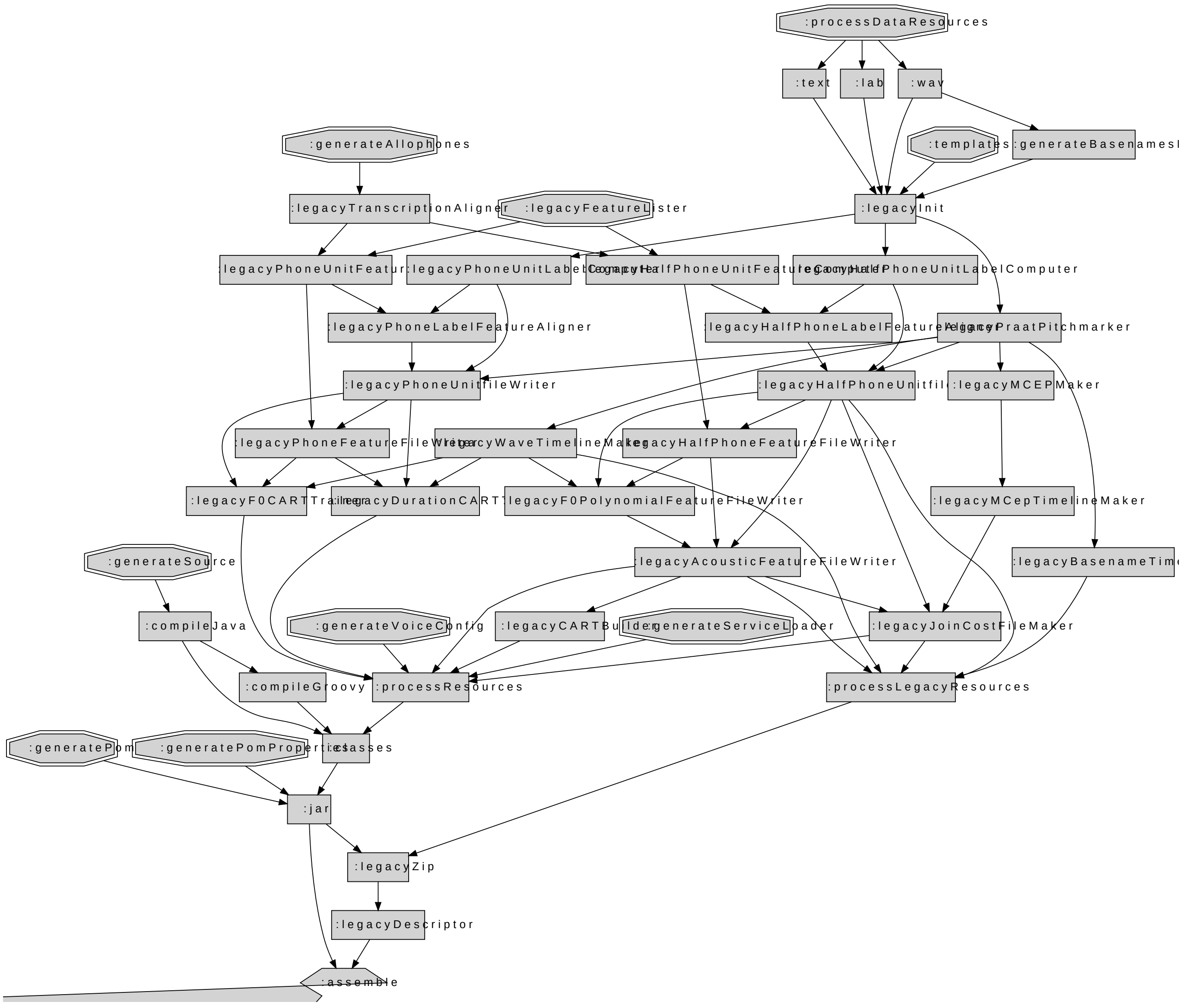
Now you can go to you local server web-interface in your browser and try voice *fraser en_US male unitselection general*

So, if everything is good, *welcome to the world of speech synthesis!*

What is under the hood?

First of all, the goal of the seminar on a high-level was to build the voice using MaryTTS. MaryTTS is responsible for building the voice model, calculate stats etc. We built the voice based on **Unit Selection**.

Let's have a look at the Mary task *execution graph* first



The top right corner is our start, left down is the end.

In order to build a voice we need to have three types of data:

1. *text* (text prompts, what a speaker records), e.g. text
2. *wav* (the recording itself), e.g. can be downloaded here
3. *lab* (annotated file), respective example here

How to get *text*?

We used the 'arctic' corpus to generate around 1000 sentences, which were saved to individual .txt files. These cover at least all the diphones. We used them as prompts for the voice recording.

How to get *wav*?

We had a speaker read out the sentences, and recorded them. We used praat to segment the voice recordings into individual .wav files.

How to get *lab*?

We used the auto-alignment tool from Munich University to generate label files which, for each utterance, tagged each phoneme with its corresponding SAMPA symbol. We converted the output TextGrid files into label files using Praat.

What we did in detail

1. Compiled a list of ‘arctic’ prompts for recording
2. Recorded Fraser's voice reading out the prompts and got three files in wav format (the first being the utterances themselves, the second being beeps between prompts, and the third being midi-whistles used to quickly find utterances with errors). All files were saved to .flac format.
3. Using Praat GUI, we manually removed certain segments using MIDI channel (skipping repetitions and badly recorded sentences), and got 893 utterances in the end.
4. Segmented voice using beeps (Created a WAV file for every valid segment using a praat script)
5. Synced filenames between WAV and TXT files using a python script (as a preparation for MAUS)
6. Fed the WAV and TXT files into MAUS, got output (alignment, in the format of TextGrids) from MAUS. Turned the resulting TextGrid files into label (lab) files using a praat script.
7. Used a voice template (gradle script) to build a voice using the WAV, TXT and LAB files (report problem with en_GB segmentation, thus we switched to en_US and run the MAUS part again) -> Resegmented the data using American English, fo that we had to change the phoneme labels given by MAUS to match the ones read by marytts using a python script
8. Used ./gradlew run command to run a MaryTTS instance with the new language

Note that to be able to build everything praat should be installed (for MacOS find solution here) On OSX during the seminar we faced some installation issues of praat, because the working script was edited right before we started, but in the end we managed to fix everything manually (providing correct hyperlinks), but now it should work without an issue.

1. Compile a list of 'arctic prompts' for recording

As we wanted to build the voice with the best quality possible, we needed to provide a good coverage of diphones. So we used the freely available 'arctic' text snippets , compiled from public data. This automatically created a variety of files, including a directory of text files containing a selection of text snippets from various books. The snippets together had an appropriate spread of diphone coverage.

We needed to convert the data into a convenient form for the speaker to read out from. We followed the instructions to generate some prompts in PDF form that would pop up on the screen one after the other.

We faced some environmental issues on OSX and Ubuntu, but resolved them succussfully.

2. Recording the voice

We were lucky to have a 2.5h session in a recording studio. The studio had an anechoic booth suitable for TTS voice recordings. We were a team of three. It was important that the voice remained constant throughout the recording, so we took regular breaks.

1. 'Native *British* English speaker', who would read out the prompts one after the other - *Fraser*
2. 'Prompts-master', the person responsible for showing a new prompt after previous one is read and recorded - *Nataniel*
3. 'Editor', the person responsible for marking each recording with special midi-beeps to speed up the process of extracting individual utterances later - *Yauhen*

Three audio tracks were recorded to .flac format: the utterances themselves, the beeps between prompts, and the midi-whistles used to quickly find utterances with errors. We loaded the files into Praat, and used it to create TextGrid files which segmented the beep and midi-whistle sounds into where there is sound, and where it is silent.

3. Removing the errors

As a group, we manually went through each segment (separated by the beeps), and evaluated if what the speaker said matched the appropriate prompt. Speaking non-stop for 2.5 hours means that mistakes are inevitably made, and the safest way to remove these mistakes was to do it manually. Seeing as we only recording around 900 utterances, it was feasible. At every point where a valid utterance was spoken, we left the label of the segment in the beep TextGrid file as "voice", but for all problematic utterances, we removed the label entirely. Later on, only segments with the label "voice" will be used. We also had to manually adjust the boundaries of some utterances, as there was confusion during the recording when the speaker repeated a failed utterance, and the prompts-master made the beep sound at the wrong time.

4. Segmenting the voice

We had one large TextGrid file, which contained one important tier: The beep tier, which had a segment with the label "voice" for every valid utterance. Labels which weren’t “voice” were ignored later. This includes the manually annotated errors - any “voice” labels for problematic sentences were removed.

In order to build our voice using MaryTTS, we needed three directories containing the following: A list of .txt files, each containing the words read out by the speaker A list of .wav files, each containing the audio of each utterance A list of .lab files, a special type of specification for each utterance which lists the end times of each phoneme and their appropriate SAMPA label

We obtained the .wav files using a praat script called split_in_wavs_based_on_beeps.praat

The important part of the script is this part:

```
for interval from start_from to end_at
    select TextGrid 'gridname$'
    intname$ = ""
    intname$ = Get label of interval... tier interval
    intervalstart = Get starting point... tier interval
    if intervalstart > margin
        intervalstart = intervalstart - margin
    else
        intervalstart = 0
    endif
    intervalend = Get end point... tier interval
    if intervalend < endoffile - margin
        intervalend = intervalend + margin
    else
        intervalend = endoffile
    endif
    select LongSound 'soundname$'
    Extract part... intervalstart intervalend no
    filename$ = intname$
    intervalfile$ = "'folder$'" + "'prefix$'" + "'filename$'" + "'suffix$'" + ".wav"
    indexnumber = 0
    while fileReadable (intervalfile$)
    indexnumber = indexnumber + 1
        intervalfile$ = "'folder$'" + "'prefix$'" + "'filename$'" +
"'suffix$''indexnumber'" + ".wav"
    endwhile
    Write to WAV file... 'intervalfile$'
    Remove
endfor
```

This part loops through each utterance and calculates its beginning and ending times, and writes them to a .wav file using the label on a separate tier. After each sound object is created, and written to file, it is then deleted, in order to save space.

With the sound opened as a LongSound object, and the TextGrid with all the utterances labeled after the names of the arctic prompts, we were able to execute the praat script. The script filled the /wav directory with .wav files named after each prompt. The segmenter that we used afterwards required that each .wav file and .txt file had the same names. In order to check that each file aligned with one another, we listened to a few of the audio files and checked to see if the text file matched. Two or three files caused the files to be misaligned, due to oversights in our manual check for errors. We went back, removed these examples, and reran the scripts, and then had our aligned data.

The /wav and /txt directories served as input into the Munich Automatic Segmenter System (MAUS).

This is an external software that we used to help generate the .lab files. Using a .txt file containing the words that were said in the corresponding .wav file, the segmenter generates a TextGrid file with intervals for each phoneme and their label, based on what the system’s speech recognition system could hear. We used the web interface to upload 100 files at a time, until we had 892 TextGrid files containing every phoneme from each utterance.

We used a simple praat script to convert the TextGrid files into label files:

```
select Strings listOfFiles
numberOfFiles = Get number of strings
for index to numberOfFiles
    select Strings listOfFiles
    filename$ = Get string... index
    Read from file... 'filename$'
    newobject = Extract tier... 1
    Save as Xwaves label file... text/'filename$'
    Remove
endfor
```

The script loops through each TextGrid file by referring to a Strings object in praat. The String object is just a list of filenames gotten from the arctic prompts. From this the script can open the relevant TextGrid file, extract the useful tier containing the phonemic information, and save it as a label file with an equivalent filename.

At this point we had the /wav, /txt and /lab files, and were ready to build the voice.

The voice was built using the "voice-template" project provided by the lecturer. This template project relies on the "de.dfki.mary.voicebuilding-legacy" plugin in order to process the /wav, /txt and /lab files and make them available for MaryTTS.

The project's build.gradle file was configured with a "voice" object, initially to a british male speaker. It was then built using the "gradlew assemble" command. During this first compilation attempt divergences emerged between the SAMPA notations used by MaryTTS and MAUS, resulting in the compilation failing.

When the lab files were being translated we identified an issue on MaryTTS. Despite its theoretical support for British English (en_GB), the list of valid allophones was a copy of the American English (en_US) one, i.e., allophones specific to the British pronounciation were not available, rendering the file translation impossible.

Due to this issue, the /wav and /txt files were once again segmented with MAUS, this time using American English, and praat was again used to convert the resulting TextGrid files into \lab ones.

After the files were converted, the "voice-template" project was adapted, in order to use American English, and the build process was executed. In this second compilation attempt, divergences between the notations used by MaryTTS and MAUS emerged once again, this time however, it was possible to map the phonetic symbols to suit MaryTTS.

In order to translate the symbols used by MAUS to symbols recognized by MaryTTS a python script was written:

```
import sys
from os import listdir
from os.path import isfile, join

symbols = [
    ['i:','i'], ['u:','u'], ['aI','AI'], ['3`','r='], ['eI','EI'],
    ['<p:>','_'], ['?' ,'_'], ['O:','O'], ['R' ,'r'], ['Q','A'], ['h-','h'],
    ['6','r='], ['4','d'], ['I@','I'], ['<usb>','_'], ['m=','r'], ['l=','l'],
    ['n=','n'], ['a:', 'a'], ['a', 'A'], ['o:', 'O'], ['y:', 'I'], ['e', 'E'],
    ['e:', 'E'], ['E:', 'E'], ['OY', 'OI'], ['y', 'I'], ['o', 'O'], ['9', 'V'],
    ['x', 'k'], ['C', 'S'], ['Y', 'I'], ['2:', 'r='] ]
```

```
labDirectory = sys.argv[1]
outputDirectory = sys.argv[2]

labFiles = [f for f in listdir(labDirectory) if isfile(join(labDirectory, f))]
labFiles.sort()

for lab in labFiles:
    print('Processing file %s' % lab)
    newFileName = join(outputDirectory, lab)
    f = open(join(labDirectory, lab), 'r')
    content = f.readlines()

    i = 0
    while i < len(content):
        for symbol in symbols:
            originalSymbol = symbol[0]
            correctSymbol = symbol[1]
            content[i] = content[i].replace(originalSymbol + '\n', correctSymbol +
'\n')

        i += 1
    f.close()
    f = open(newFileName, 'w')
    f.writelines(content)
    f.close()
```

This script receives two command line parameters: an input directory, where the original /lab files are stored, and an output directory, where the new files will be stored. It then loads each file from the output directory, replaces the invalid symbols and saves a new file in the output directory.

The new files were used on the "voice-template" project and the "gradlew assemble" command was executed. This time the project was correctly built and the new voice was ready to be used.

To test the newly built voice the command "gradlew run" was used and as testing, some random text segments were extracted from british (www.theguardian.co.uk) and american (www.cnn.com) news websites.

Find the short test recording here