A Flexible Front-End for HTS

Matthew P. Aylett[12], *Rasmus Dall*[2], *Arnab Ghoshal*[2], *Gustav Eje Henter*[2], *Thomas Merritt*[2]

[1]CereProc Ltd., U.K.
[2]CSTR, University of Edinburgh, U.K.
matthewa@inf.ed.ac.uk

Abstract

Parametric speech synthesis techniques depend on full context acoustic models generated by language front-ends, which analyse linguistic and phonetic structure. HTS, the leading parametric synthesis system, can use a number of different front-ends to generate full context models for synthesis and training. In this paper we explore the use of a new text processing front-end that has been added to the speech recognition toolkit Kaldi as part of an ongoing project to produce a new parametric speech synthesis system, Idlak. The use of XML specification files, a modular design, and modern coding and testing approaches, make the Idlak front-end ideal for adding, altering and experimenting with the contexts used in full context acoustic models. The Idlak front-end was evaluated against the standard Festival front-end in the HTS system. Results from the Idlak front-end compare well with the more mature Festival front-end (Idlak - 2.83 MOS vs Festival - 2.85 MOS), although a slight reduction in naturalness perceived by non-native English speakers can be attributed to Festival's insertion of non-punctuated pauses.

Index Terms: speech synthesis, text processing, parametric synthesis, Kaldi, Idlak

10.21437/Interspeech.2014-320

1. Introduction

Hidden Markov Model (HMM)-based synthesis has been shown to be effective in synthesising speech. HTS (H Triple S - Hidden Markov Model Speech Synthesis System) is the dominant open source system, and was pioneered at Tokuda's group at Nagoya over a decade ago. The current system is mature and produces leading results in synthesis based on parametric modelling [1]. In speech synthesis, the task of analysing text to decide pronunciation and linguistic structure is often termed the front-end. The process of generating waveforms from this analysed text is then the back-end. HTS is a back-end system only, and depends on 3rd party systems to carry out the initial stage of processing. Whereas significant research has been devoted to the back-end of parametric speech synthesis, relatively little work has looked at the effect of the contexts chosen and the algorithms used to produce their values in the front-end of a parametric system. Lu and King [2] examined the use of Bayesian approaches to selection and factorisation of context features, but in general the selection of the context features tends to default to those suggested in HTS publications (e.g. [3]), and the decision tree process is assumed to deal with any extraneous contexts.

Research focusing on the acoustic model context architecture using HTS is currently hampered by practical considerations. Firstly, it requires familiarity with 3rd party system front-ends, some of which have been developed over many years and are hard to modify without considerable investment of time and energy; secondly, a requirement that the question set used to build the decision trees matches the feature set used. These question set files are generally written by hand, so modifying features is a painstaking and error prone operation.

Idlak is a project to build an end-to-end parametric synthesis system within Kaldi [4] – a liberally licensed automatic speech recognition (ASR) toolkit. As part of Idlak, a front-end that generates full context models compatible with HTS has been developed. Based on a modular design, modern coding and testing approaches, and with XML as a main interface, this front-end aims to simplify the front-end process for parametric synthesis while still producing state-of-the-art results. In this paper we present the design decisions made to achieve this, and an evaluation of this new front-end using a standard, publicly available, HTS-demo. All the results reported here can be reproduced by other research groups, and we hope that the availability of the Idlak front-end will encourage research into the context extraction part of parametric synthesis, as well as the development of Idlak front-ends for other languages.

The remainder of the paper is structured as follows: Section 2 describes the goals and design of Idlak. Section 3 describes how Idlak was adapted to the context extraction requirements of HTS. Section 4 provides an experimental evaluation of the new Idlak HTS front-end, with results discussed in section 5, while section 6 concludes.

2. Design of Idlak

2.1. Background

Kaldi is an open source speech recognition toolkit available under the Apache 2.0 license. Kaldi's objective was to produce a modern flexible code base in C++, which is easy to understand, modify and extend. The system compiles on Unix-like systems and on Microsoft Windows and is available from SourceForge (see http://kaldi.sf.net/).

As Kaldi was created for ASR there is potential for cross over between research in ASR and parametric text-to-speech (TTS). In addition, parametric TTS also offers a means for building speech synthesis systems in many languages and accents where full unit selection systems are not commercially viable. However the currently dominant parametric TTS system (HTS) has a number of limitations:

- Licensing based on HTK and patches prevents open development and limits commercial exploitation
- 3rd party packages are required for front-end processing and the state-of-the-art systems are dependent on restrictively licensed packages (combilex [5], STRAIGHT [6])
- Although mature and with excellent quality, it is hard to create new voices or languages for the speech synthesis novice as well as the ASR professional

```
<?xml version="1.0"?>
<parent>
        <utt uttid="1" no_phrases="2">
                <spt phraseid="1" no_wrds="2">
                        <ws col="0">
</ws>
                        <break type="4" time="0.011" />
                        <tk norm="hello" lc="true" uc="true" pos="NN" posset="1"
                           wordid="1" pron="hh ah0 l ow1"
                           spron="hh+ah0|l+ow1|" nosyl="2">
                                Hello
                                <ws> </ws>
                                <syl val="hh+ah0" stress="0" sylid="1" nophons="2">
                                        <phon val="hh" type="onset" phonid="1" />
                                        <phon val="ah" type="nucleus" phonid="2" />
                                </syl>
                                <syl val="l+ow1" stress="1" sylid="2" nophons="2">
                                        <phon val="l" type="onset" phonid="1" />
                                        <phon val="ow" type="nucleus" phonid="2" />
                                </syl>
                        </tk>
          ...
```

Figure 1: *Example XML output from the Idlak front-end for the word "Hello" in the sentence "Hello there, one two three"*

The Kaldi speech synthesis project (Idlak) addresses these issues by offering:

- A simple permissive licensing environment
- A close connection with state-of-the-art ASR techniques through its integration with Kaldi
- An end-to-end system for both creating and synthesising from parametric TTS voices

2.2. Architecture

Kaldi is based on a series of command line programs which can operate on groups of files of various types. The Idlak front-end follows this approach but introduces a new file format to Kaldi, in the form of XML. A number of different components make up the Idlak front-end:

2.2.1. Text Processing Modules

Written to the Google coding standard[1] and integrated into the Kaldi codebase, the Idlak text processing modules each operate on an XML marked-up stream of text. Each module will typically add structure to the XML and may be dependent on structure added by previous modules. For example, a syllabify module requires a pronunciation generated by a pronounce module (Figure 1 shows an example of some XML output from the syllabify module). The modules can be chained into command line binaries that use Kaldi-style options, and can take input from pipes or files. Currently two command line programs are in the system: idlaktxp which carries out text processing, and idlakcex, which takes output from idlaktxp and adds context model names in either a Kaldi or an HTS format. Figure 2, shows the current modules that form idlaktxp.

2.2.2. Text Processing Database (TPDB)

A number of XML files which contain the linguistic and phonetic information required for the text processing modules. For example, a lexicon built from the open source CMU General American lexicon[2], letter to sound rules built from this lexicon, files which determine UTF8 handling, files describing how punctuation affects pausing etc. The files are structured by language (an ISO 2 letter code), accent (a non standard two letter code), speaker (a non standard 3 letter code) and region (an ISO 2 letter code). A synthetic voice would contain a set of these XML data files customised for a specific voice, whereas a General American grapheme to phoneme system would contain speaker independent data files.

[1] https://code.google.com/p/google-styleguide/

[2] http://www.speech.cs.cmu.edu/cgi-bin/cmudict

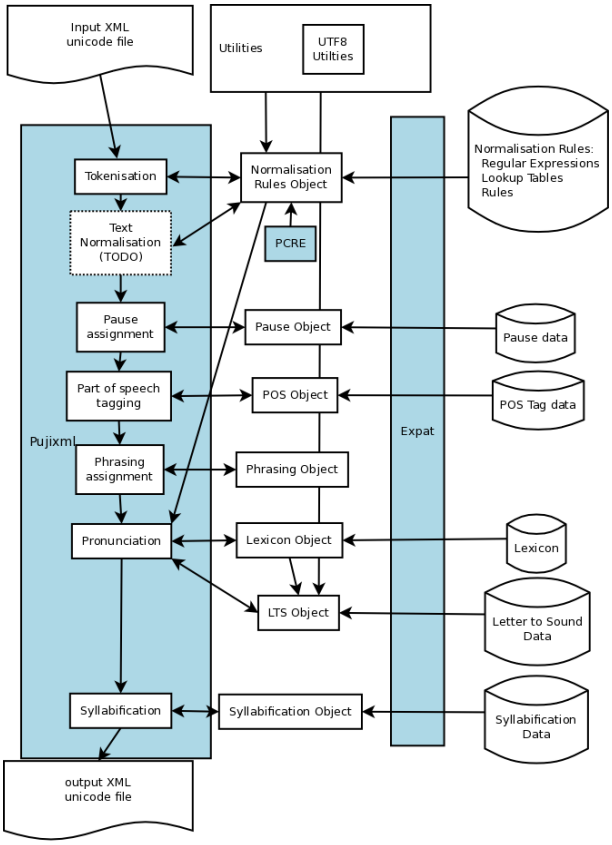

Figure 2: *Idlak text processing system (idlaktxp). The system comprises of a set of modules operating on XML input and producing further tagged XML.*

2.2.3. Voice Building System

In contrast to ASR, TTS systems are very dependent on text normalisation (for instance the conversion of dates and currency formats), which introduces a dependence between the front-end and models built for the back-end. The voice building system in Idlak is a python-based framework with clear input and output specifications.

1. Customise a general text processing system to a specific voice and voice database (for example duplicate speaker specific pausing and pronunciation).
2. Allow an audit trail, so that errors in the final synthesis can be tracked back to the voice building step that caused them.
3. Allow a single-click modular build process.

Figure 3 shows the structure of the Idlak voice building system. In order to generate appropriate front-end output for HTS a bespoke voice building module (hts_test) was added which was dependent on the alignment and context extraction modules.

3. Idlak Model Context Extraction for HTS

In HTS, contexts are specified in the full acoustic model name for each phone. In the default system, 43 contexts over a number of linguistic and phonetic domains (6 phone, 18 syllable, 11 word, 5 phrase, 3 utterance) are used. These vary from contexts such as quinphone context, the number of syllables since the last stressed syllable, part of speech, to phrasing information

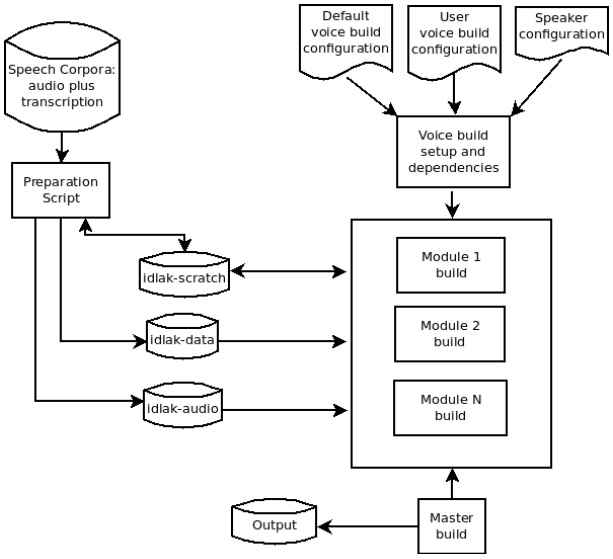

Figure 3: *The python-based voice building architecture of Idlak, capable of single-click end-to-end voice building.*

etc. Each context is either a string (such as a phone name) or an integer (such as the number of syllables in a phrase). In order to produce a model name for HTK these contexts are concatenated with a delimiter between each context. Below is a full context model name for a schwa (*ax* in the second syllable of the word "Alice") generated by Festival for the HTS-demo (line breaks and indentation are added for readability).

```
aeˆl-ax+s=w@2_2/A:1_1_1/B:0-0-3@2-1&2-21
#1-12$1-3!1-1;1-7|ax/C:1+0+3/D:0_0
/E:content+2@1+15&1+7#0+2/F:aux_1
/G:0_0/H:22=15ˆ1=1|NONE/I:0=0/J:22+15-1
```

In HTS there is a critical dependency between the final decision trees, the question set and the front-end. Furthermore, in HTK decision tree questions are based on regular expressions which means a poor use of delimiters can cause further errors. Different choices in how a specific context is extracted, what a NULL value may be, and the sequence of contexts will all cause failures if not consistent across all parts of the system.

In Idlak a unique context name is used to tie the function which extracts the context from the XML output of idlaktxp, the type, the delimiter and the questions that can be asked about it. To further increase the flexibility a lookup table can be added to map output from the extraction function to alternative symbols (for example making it easy to swap 'sil' for 'pau' if required). For a specific voice, a set of contexts are selected and the system ensures they remain consistent with the question set, the model building stage and the synthesis stage.

Take for example the number of phones in the current syllable. In Idlak, the unique name of this context is SyllableNumPhones, it is an Integer context, and the extraction function will search for the *nophons* attribute in the XML *phon* tag and return the value. The entry in the XML context architecture, which can be customised by users without requiring code changes, is:

```
<feat
   delim="/09:"
   htsname="C-Syl_Num-Segs"
   desc="current_syllable_no_segments"
   name="SyllableNumPhones"
   min="0"
   max="7"/>
```

Where *delim* is the delimiter to use before the context value in the full context model name, *htsname* is the HTS feature name from the HTS question set, *desc* is a description, and *min,max* is a range for an integer valued context. This combines with the entry in the question specification file shown below:

```
<feat htsname="C-Syl_Num-Segs"
      name="SyllableNumPhones">
         <qs name="C-Syl_Num-Segs==0">
               0
         </qs>
          ...
         <qs name="C-Syl_Num-Segs&lt;=1">
               0 1
         </qs>
          ...
 </feat>
```

An example model for the same phone produced by this system is shown below (line breaks and indentation are added for readability). Using the CMU lexicon and LTS rules built upon it the phone from the second syllable of "Alice" is represented by an unstressed front low vowel (ah). Apart from phone context which uses HTK style delimiters, all other delimiters are in the form */nn:* where *nn* is a two digit number.

```
ˆpau˜ae-l+ah=s/00:0/01:2/02:1/03:0/04:1/05:PAU
/06:NN/07:VBD/08:1/09:3/10:3/11:0/12:2/13:1
/14:15/17:LL
```

In Idlak, we have begun with a conservative set of 21 features (7 phone, 6 syllable, 6 word, 2 phrase, 0 utterance). However the architecture makes it easy to add more if required.

In the current HTS system, a single text file containing two sentences will produce a different quantity, and type, of models, compared with synthesising each sentence in its own file. This is because the silence model between the sentences will not be the same as the two silence models ending file one and starting file two. To avoid this problem, in Idlak, each model file represents a phrase (defined as speech separated by silence) rather than an utterance or sentence. There are thus *always* two silence models between all phrases, one completing the previous phrase and one starting the subsequent phrase.

4. Evaluation

In order to evaluate the differences between the Idlak and Festival front-ends, the Idlak front-end was incorporated into the the HTS demo[3]. The current HTS demo does not generate test models using Festival but includes the acoustic models as part of the distribution. As it was unclear which version of Festival generated these models, models were also recreated using the Festival system downloaded for the HTS demo. The test sentences in the HTS demo are from the initial paragraphs of Lewis Carroll's "Alice in Wonderland".

Next, Idlak was used to generate model files by phrase, based on the Arctic 16 kHz release of SLT (the same voice data used in the HTS demo) and a question file for building the decision trees. In addition, timing information from the Idlak alignment was used to replace the 48 kHz voice data audio in the HTS demo with 48kHz data cut up by phrase rather than recorded utterance.

20 random phrases, between 6 to 20 words in length, were chosen from the output synthesised by the HTS demo. Speech matching these word sequences with a matching start and end pause was also taken from output generated from test acoustic models found in the HTS demo distribution (block 1), and models generated from scratch using Festival (block 2). Currently Idlak will only insert pauses based on punctuation, whereas the

[3]http://hts.sp.nitech.ac.jp/archives/2.3alpha/HTS-demo_CMU-ARCTIC-SLT.tar.bz2

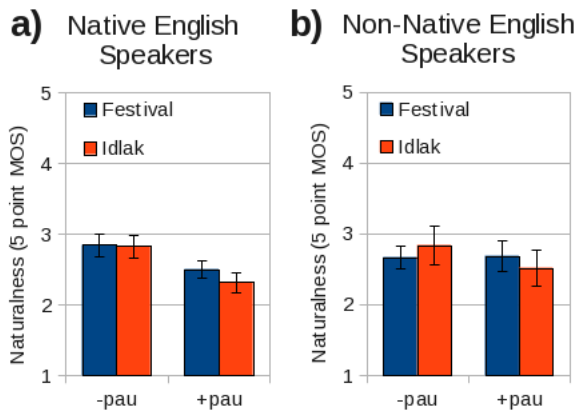

Figure 4: *Evaluation results by system and pause grouping. –pau - neither Idlak not Festival inserted pauses into the phrase. +pau - Festival inserted between 1 and 2 pauses in the phrase whereas Idlak output was pause free.*

SLT voice distributed by CMU, and used with Festival, will also insert pauses based on n-grams trained on pausing and part of speech tagging.

33 subjects with normal hearing (25 native English speakers and 8 non-native English speakers with a very high level of spoken English) were allocated to each block (15 to block 1 and 18 to block 2). They listened, over headphones, to each synthesised section of speech based on Festival and Idlak models, and were asked "How natural is the audio?", responding on a 5 point scale (Bad, Poor, Fair, Good, Excellent).

Informal feedback suggested that non-punctuated pause insertion had a positive impact on perceptions of naturalness. e.g. "I find that I like the [synthesised speech] more pausing generally – however sometimes it ruins the sentence quite a bit when the pauses are too long (or misplaced)."

4.1. Analysis

In order to take into account the effect of inserted pauses, sentences from both blocks were split into two groups, those where Festival and Idlak had the same pause structure (12), and those where the pause structure was different (28). A mean opinion score (MOS) was calculated for each group by subject.

A by-subjects, repeated measures ANOVA was carried out with front-end type and pause difference (+/-pau) as nested variables, and with block and subject nativeness as grouping variables. An MOS-based ANOVA is acceptable based on the central limit theorem since each cell has at least ten data points. Neither block nor nativeness has a significant effect on MOS scores, however +/-pau had a substantial impact on the results ($F(1,29) = 11.616$, $p<0.005$). This drop in MOS across *both* systems cannot be attributed to pause insertion, as the Idlak stimuli matching the Festival +pau did not contain pauses. Rather we can attribute an overall drop in MOS for the +pau condition to a longer phrase length (mean word length 13.5 +pau vs 8.4 -pau). To support the informal feedback, that Festival pause insertion improved naturalness, we would have expected a significant interaction between +/-pau and system which was no the case.

However, although nativeness did not have a significant effect on MOS, when separate ANOVAS were carried out on native and non-native subjects a different reaction to inserted pauses appears for non-native subjects. As with the combined analysis, native subjects show a significant effect for +/-pau ($F(1,23)=39.779$, $p<0.001$), whereas non-native subjects show a weakly significant result for a +/-pau insertion with system ($F(1,6) = 6.653$, $p<0.05$); see Figure 4. Therefor there is marginal evidence that non-punctuated pause insertion improved naturalness for HTS synthesis.

5. Discussion

Overall the Idlak front-end performed well, producing very similar results in terms of synthetic speech naturalness as Festival, except for Festival's insertion of non-punctuated pauses. Informal feedback suggested the pausing was an important element in the perception of naturalness but this effect was only significant for non-native speakers. However as Figure 4a shows this is also a tendency for native subjects.

The Idlak front-end currently has less than half of the contexts used by standard HTS front-ends. This is important because less contexts simplify implementation and also decision tree output. However, the results presented here cannot guarantee that more contexts *do not* improve naturalness. Given more stimuli and more subjects they may. The assumption in HTS is that, at worst, they will do nothing. Therefore although many of the contexts appear to have a marginal effect on quality, this is not an argument to remove them.

Rather as, Lu and King [2] argue, we may look more closely at the contexts used, their dependencies, to try to identify a high-performing and suitably parsimonious set of contexts. Using a very large set of contexts is not without substantial engineering cost in terms of context extraction code, data to support a context, and calculating the context for input text.

Idlak offers a flexible and efficient means of examining, in more depth, the interaction between contexts used in full models and their effect on speech synthesis quality.

6. Conclusion

There was no evidence that the phrase based (rather than utterance based) approach to modelling pausing used by Idlak caused any degradation in quality. In addition the 21 features currently used by the Idlak front-end performed well and produce comparative results with the HTS demo using Festival. Results from this work also suggest that non-punctuated pause insertion may yet be able to deliver improvements in speech synthesis quality, especially in parametric systems.

Future work hopes to extend Idlak's front-end to additional languages and use the flexible context architecture in the system as a means of exploring the effect of context choice on speech synthesis quality. The data evaluated in this work can be generated from open source downloads and instructions for generating this test can be found in the Idlak documentation within Kaldi[4].

The next stage in Idlak will be to use the front-end to build the decision trees and full context acoustic models and compare this output against baseline HTS demo output.

7. Acknowledgements

This work was funded by the Royal Society through a Royal Society Industrial Fellowship and by the EPSRC Programme Grant EP/I031022/1 (Natural Speech Technology), also thanks to Richard Williams for code refactoring and development.

[4]Currently the Idlak branch of Kaldi can be installed with `svn co https://svn.code.sf.net/p/kaldi/code/sandbox/idlak`

8. References

[1] H. Zen, T. Nose, J. Yamagishi, S. Sako, T. Masuko, A. Black, and K. Tokuda, "The HMM-based speech synthesis system (HTS) version 2.0," in *Proc. of Sixth ISCA Workshop on Speech Synthesis*, 2007, pp. 294–299.

[2] H. Lu and S. King, "Using Bayesian networks to find relevant context features for HMM-based speech synthesis." in *INTERSPEECH*, 2012.

[3] K. Tokuda, Y. Nankaku, T. Toda, H. Zen, J. Yamagishi, and K. Oura, "Speech synthesis based on hidden Markov models," *Proceedings of the IEEE*, vol. 101, no. 5, pp. 1234–52, 2013.

[4] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz *et al.*, "The Kaldi speech recognition toolkit," in *Proc. ASRU*, 2011.

[5] S. Fitt and K. Richmond, "Redundancy and productivity in the speech technology lexicon-can we do better?" in *INTERSPEECH*, 2006.

[6] H. Kawahara, J. Estill, and O. Fujimura, "Aperiodicity extraction and control using mixed mode excitation and group delay manipulation for a high quality speech analysis, modification and synthesis system STRAIGHT," *Proc. MAVEBA*, pp. 13–15, 2001.