IDIAP COMMUNICATION REPORT

INTEGRATION OF REAL-TIME SPEECH PROCESSING TECHNOLOGIES FOR ONLINE GAMING

Dhananjay Ram Petr Motlicek Blaise Potard

Idiap-Com-01-2016

FEBRUARY 2016

Centre du Parc, Rue Marconi 19, P.O. Box 592, CH - 1920 Martigny
T +41 27 721 77 11 F +41 27 721 77 12 info@idiap.ch www.idiap.ch

Integration of Real-Time Speech Processing Technologies for Online Gaming

Dhananjay Ram, Petr Motlicek, Blaise Potard
Idiap Research Institute, Martigny, Switzerland
email: {dram, motlicek, potard}@idiap.ch

April 1, 2015

Abstract

This work demonstrates an application of different real-time speech technologies, exploited in an online gaming scenario. The game developed for this purpose is inspired by the famous television based quiz-game show, "Who wants to be a millionaire", in which multiple-choice questions of increasing difficulty are asked to the participant. Text-to-speech synthesis is used to read out the questions and the possible answers to the user, while an automatic speech recognition engine is exploited to get input from the player, in order to proceed through the game. The speech data is recorded from the user with the help of a real-time voice activity detector to select speech segments from the input audio data. The developed Java application allows an automatic insertion of new multiple-choice questions, of different complexity, which could then be selected during the game.

1 Introduction

The work is a demonstration of two real-time speech technologies, namely Automatic Speech Recognition (ASR) and Text-to-Speech Synthesis (TTS), through the implementation of an online game. The game simulates the famous television based quiz-game show called, "Who wants to be a millionaire". This is a very simple quiz game in which users are presented with questions, along with 4 possible choices, at each stage of the game. Among the four options, only one answer is correct.

In our game, the question-answers are displayed, as well as pronounced to the player using synthetic speech, to make the game more interactive. After this, the player needs to speak out the answer in order to proceed through the game. If the answer is correct, the player is allowed to move to the next difficulty level of the game, otherwise, he/she loses. To help players to proceed through the game, three types of assistance are normally available in the TV game, namely, "Fifty Fifty", "Call Friend" and "Help of the Auditory". Only the "Fifty Fifty" option is fully implemented in the current version of the game.

The work presented in this report is a follow-up work on an earlier version of this game. Readers are advised to consult the reports on earlier work [1, 2] before proceeding further with this report. In the earlier version, the speech input did not use voice activity detection, instead recordings audio segments of fixed durations at fixed intervals of time: the player was required to wait for a signal from the game to start speaking, and the game would only start processing the player input after the whole segment (4s long) was recorded. The player had to wait accordingly, even if the speech segment was very short. In addition, the player had to wait further while the speech segment was being recognised. In order to overcome these shortcomings, a real-time voice activity detector working in online mode is now implemented, so that the player can provide input speech any-time, and the subsequent ASR engine will start processing the data with a very small delay after the player starts speaking, with the aim of providing recognised input as soon as possible after the player stops speaking.

Furthermore, in the earlier version of the game, an existing HTK [3] based ASR system was exploited to perform Key-Word Spotting (KWS). In this version of the game, a new KWS system based on the Kaldi [4] speech recognition toolkit was devised.

The report is organized as follows. A brief revisit to the architecture of the game is presented in the following section. This is followed by descriptions of the client-side and server-side implementation of the game. Finally, the work is concluded in the last section with a summary of contributions, and directions for future work.

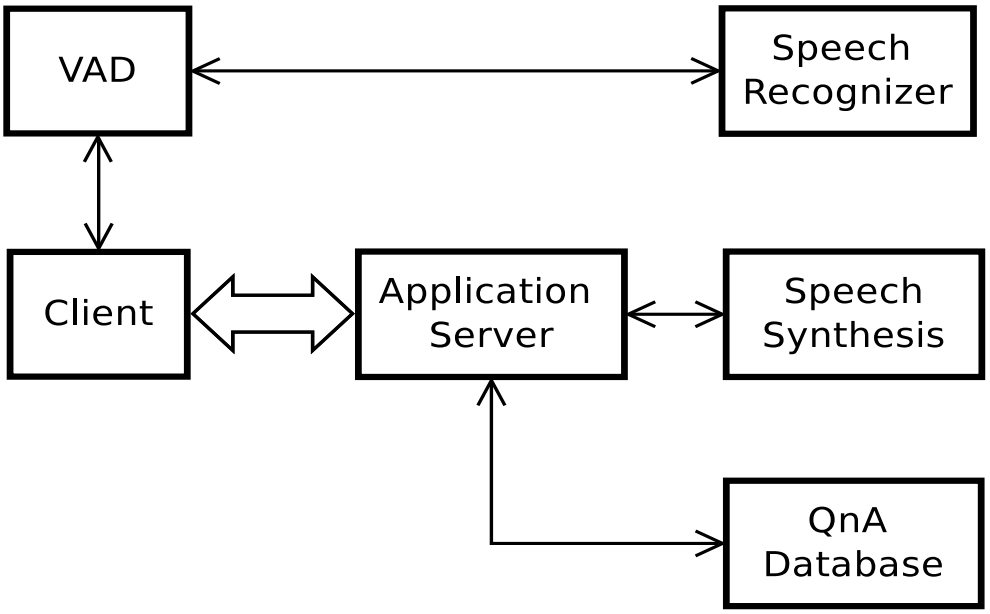

Figure 1: Block diagram of the game implementation

2 Architecture

The game follows a client-server architecture. The client implements a gaming (graphical) interface for the player. The server-side can operate on a different machine, or in the background on the same machine, and incorporate several services.

While playing the game, the client records input speech from the player with the help of a voice activity detector (VAD) and transmits the recorded data to the server-side for further processing. The client also interacts with the server to obtain the multi-choice questions depending on the state of the game. Finally, it receives synthesized speech generated by a TTS engine on the server to read-out the question-answers to the player.

The most time-critical operation (ASR) is implemented through an ad-hoc streaming interface developed in the context of the D-Box project: the client streams audio data in which voice activity has been detected, and the server returns ASR results at the end of the active segments. The two other services (questions and TTS) are implemented as Web Services and written in java.

3 The Client

This section describes the gaming interface and briefly introduces different functions of the client. This is followed by a detailed discussion about the

VAD integrated into the game.

3.1 The Interface

In the earlier version of the game, there was a button implemented for the player to indicate when the speech is being recorded from the microphone. The color of the button (i.e., green/red) indicated the time period to start/stop speaking. Speech data from the player was recorded for a constant duration and processed afterward. This implementation was obviously not well suited for the players, as it restricted them from providing input at their convenience. The player was also forced to wait a significant time even if the input speech he wanted to provide was of very short duration (due to this fixed duration constraint).

In the new updated version of the game, real-time VAD is implemented, which eliminate the need for this indicator button. This VAD can segment the input audio data according to the speech/non-speech parts and filter out the non-speech segments. In this way, only speech data is further processed by the ASR component, and players are provided with a more responsive and convenient way of interacting with the game. Furthermore, this VAD uses a streaming interface, i.e. starts forwarding audio data to the ASR server before the end of an active segment, which ensures a low latency for the ASR processing.

The rest of the options have remained unchanged.

3.2 Functions

The functions of the client can be categorized into three parts, as discussed below:

- The first part implements a http request to the question-answer database to obtain a question of a specific difficulty, and its possible answers. A reply from the server is used to display the question-answers to the player.

- As a next step to proceed through the game, the client makes a request to the TTS engine with the current question-answers. The engine replies with an audio file containing generated speech corresponding to the question-answers to be played-back to the player.

- As stated above, the new version of the client applies VAD for real-time speech segmentation, and the active speech data is streamed to the server side for online ASR. Overall, the introduction of VAD on the client side allows the player to answer a question whenever he/she wishes, while the online ASR and VAD streaming mode ensure that the player receives a reaction from the game as soon as possible.

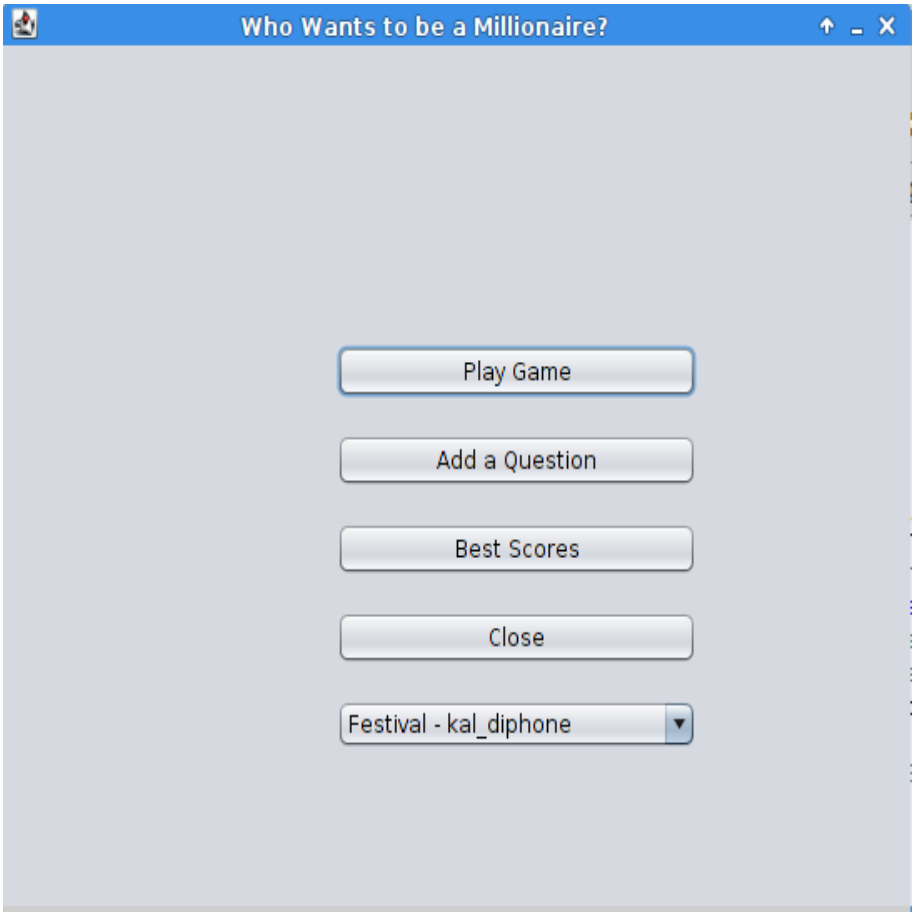

Figure 2: Welcome screen of the game.

3.3 Voice Activity Detector

This section presents a detailed description of the VAD exploited on the client side of the game application. It is inspired by a VAD implementation presented in [5]. This is a very simple VAD which can be used in real-time applications like this game. More specifically, this method proposed three different features per frame to be used for VAD:

- The first feature is the widely used short-term energy (E). Energy is the most common feature for speech/non-speech detection. Speech frames are expected to contain significantly higher amount of energy than non-speech frames. However, this feature loses its efficiency in noisy conditions, especially in lower signal-to-noise ratio (SNR). To reduce this sensitivity to noise, we compute the energy in the spectral domain, only taking account the frequency bands strongly associated with human speech. Furthermore, two other features calculated

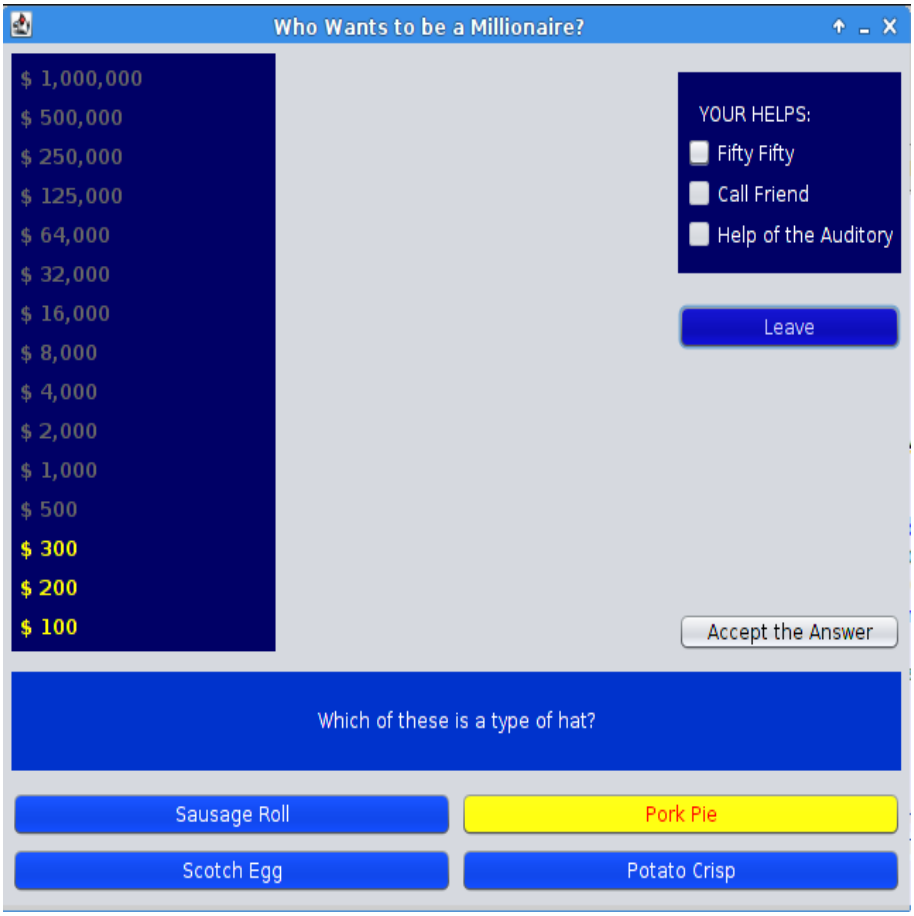

Figure 3: Main interface of the game.

in the frequency domain are also used to take a decision about the speech/non-speech status of a given frame.

- The second feature is Spectral Flatness Measure (SFM). Spectral Flatness is a measure of the noisiness of the spectrum and is a good feature in Voiced/Unvoiced/Silence detection. Generally, speech frames are less flat than non-speech frames. This feature is calculated using the following equation:

$$SFM_{db} = 10 \log_{10} \left(\frac{G.M.}{A.M.} \right), \tag{1}$$

where $A.M.$ and $G.M.$ are arithmetic and geometric means of speech spectrum respectively.

- Besides these two features, the most dominant frequency (F) component of the speech frame spectrum is used as the third feature in

order to discriminate between speech and non-speech frames. This feature is simply computed by finding the frequency corresponding to the maximum value of the spectrum magnitude. It is expected that the maximum frequency present in a speech frame cannot go beyond a certain pre-specified range, $[minF, maxF]$, corresponding to the typical frequency range of human speech. The frames having a maximum frequency beyond this range are likely to be non-speech frames.

In the VAD implemented, these three features are applied in parallel to detect the voice activity, and a frame will be considered as speech if at least two of the features judge the frame as speech. The algorithm starts with framing the audio signal (i.e., short-term analysis of speech signals). In this implementation no windowing function or overlap is applied on the frames. At the beginning of the processing, N frames are used for thresholds initialization. For each incoming speech frame, the three features are computed. The audio frame is marked as a speech frame, if at least 2 of the feature values fall over the pre-computed thresholds; otherwise it is marked as non-speech. Finally, a certain number of successive speech frames indicates the beginning of a speech segment and vice-versa. The steps to be followed are provided in the algorithm 1:

3.3.1 Evaluation of the VAD

In order to evaluate the performance of the developed VAD, experiments are performed on the speech data from the D-Box [6] project. The speech data was collected in the form of a "Wizard of Oz" simulation of a quiz game, where the player can ask any kind of question to the system, in order to guess the name of a famous character. Two English native speakers (1 male and 1 female) were acting as Wizards simulating the system's behaviour. 21 unique subjects were playing the game. The participants were undergraduates of age between 19 and 25, who are expected to be related to our ultimate target audience, and are generally non-native English speakers. 338 dialogues were collected for a total duration of 16 hours, comprising about 6.000 speaking turns. The audio data is sampled at 16kHz, and stored in RIFF format. The recorded data is manually segmented, and the speech segments are manually transcribed. The database was recorded during 6 different sessions, and these sessions were split into a train/dev/test sets, keeping only the players data (the Wizard data is also available, but is not currently exploited). The VAD is evaluated on the test data-set, which consists of recordings from two different sessions (identified by the numbers 20 and 28), with two different players (identified by the 2-letters codes pw and bp).

The performance of the developed VAD is compared with two other existing implementations. The first one is a simple energy-based VAD accessible in Shout [7]. This VAD does not have an online implementation.

Algorithm 1 An algorithm to implement VAD

Set $frame\_size$

Set one primary threshold for the first two features:

- Primary threshold for energy: $PrimE$
- Primary threshold for SFM: $PrimSFM$

Set $MinE = 0$ and $MinSFM = 0$

for i from 1 to N

    Apply FFT on each frame

    Compute frame energy, $E(i)$, over the frequency range $[minF, maxF]$.

    Compute the value of spectral flatness measure, $SFM(i)$

    If $E(i) < MinE$, then $MinE = E(i)$

    If $SFM(i) < MinSFM$, then $MinSFM = SFM(i)$

end

Set decision threshold/range for E, SFM and F as follows:

- $ThresE = PrimE \times \log(MinE)$
- $ThresSFM = PrimSFM$
- Range for F : $[minF, maxF]$

while a speech frame is available:

    Apply FFT on each frame

    Compute frame energy, $E(i)$, over the frequency range $[minF, maxF]$.

    Compute the value of spectral flatness measure, $SFM(i)$

    Find $F(i)$ as the most dominant frequency component.

    Set $Count = 0$

- If $(E(i)/MinE) \geq ThresE$, then $Count = Count + 1$
- If $(SFM(i) - MinSFM) \geq ThresSFM$, then $Count = Count + 1$
- If $minF \leq F(i) \leq maxF$, then $Count = Count + 1$

    If $Count > 1$, mark the current frame as speech, else silence.

    If current frame is silence, update $MinE$ value as,

$$MinE = \frac{(SilenceCount \times MinE) + E(i)}{SilenceCount + 1}$$

    $ThresE = PrimE \times \log(MinE)$

    n_1 successive speech frames $\rightarrow$ mark the frames as speech active.

    n_2 successive silence frames $\rightarrow$ mark the frames as speech inactive.

    if speech active and successive active frames $\geq n_3 \rightarrow$ start streaming to ASR server.

    if speech inactive and successive active frames $< n_3 \rightarrow$ drop segment.

The second one is developed by a DBOX partner company, Sikom, and implemented in the project. This commercial VAD is a very simple state-based VAD system (with three states: silence, speech active, speech pause), using time-domain energy calculation for the voice activity decision. Compared to Idiap VAD system, it is much more rudimentary: it uses a single decision criterion, computes time-domain energy, and uses a fixed threshold for voice / unvoiced decision. It can however, like our VAD system and unlike Shout, be run online.

Table 1: Comparison of performance of different VADs.

VAD method	**Database**	**Missed speech(%)**	**False alarm speech(%)**
Shout	20/pw	6.76	6.49
	28/bp	10.73	2.64
Sikom	20/pw	14.48	0.31
	28/bp	14.09	0.57
Idiap	20/pw	4.13	16.12
	28/bp	2.65	8.34

The results are evaluated by comparing the segmentation from individual VADs with manual segmentation information. The performance is indicated using a standard detection measure, indicated by the probability of missed event (i.e. a speech frame classified as non-speech) and False alarm event (i.e. a non-speech frame classified as speech). The results are shown in Table 1. Note that in this kind of applications, missed speech (i.e., number of missed speech frame) is required to be as small as possible, whereas, a greater amount of false alarm speech is not as damaging. These rates can obviously be controlled by the thresholds presented in the VAD algorithms. Since the segmented speech will be fed to the subsequent ASR engine, a higher false alarm, which indicates that the recognizer will also process non-speech data, will not largely degrade the recognition performance; however, it might lead to increased latency for ASR results. On the other hand, higher missed speech rate implies that the ASR engine will not receive all of the speech data which may largely degrade the recognition performance.

To observe the effect of the different VADs on the performance of the ASR engine, recognition experiments were performed on the DBOX data. The segmentation information from each VAD is used to evaluate its effect on recognition results. The results are shown in Table 2. Here, "Oracle VAD" refers to using the information from the manual segmentation. Recognition accuracies clearly indicate that the developed VAD from Idiap performs significantly better than the other tested VADs, and the penalty for using VAD is quite low.

Table 2: Speech recognition performance in terms of Word Error Rates (WERs) for different VAD algorithms applied.

VAD Method	**WER (%)**
Oracle VAD	25.6
Shout	38.8
Sikom	37.2
Idiap	32.0

4 The Server

The server side of the game is implemented using "apache-tomcat" [8]. It is an open-source web server and servlet container. The server needs several components, to perform different tasks. Therefore a servlet container was needed to integrate the various components of the server. In this implementation, HTTP [9] is used as the network protocol for communication between the server and the client.

Various modules of the server are discussed in the following sections.

4.1 QnA Database

The QnA database is a multilingual database containing questions and possible answers in different languages, although the present system only use the English questions set at the moment. The question-answers are accessed dynamically from a database, which is implemented using HSQLDB [10], instead of a SQL server. This saves the effort of installing an SQL server in the machine where the server resides. Each question in the database contains different fields such as questionID (an unique identifier for the question), languageID (an unique identifier for the language of the question), difficulty level, etc.

The difficulty level indicates the current stage of the game: there are normally 15 different levels, corresponding to 15 different amounts of money to be won in the TV game. As the game progresses, the difficulty level is increased by one step after each correct answer. When a query is made from the client side, it consists of languageID and difficulty level at the current stage of the game. The query returns all available questions in the database corresponding to the languageID and difficulty level. One question is then randomly selected from this set of questions and returned to the client. It contains the question, its four possible answers, the correct answer, and the questionID.

4.2 TTS Server

The Text-to-Speech (TTS) server is the second main engine operating on the server. A TTS interface is incorporated in the game to make it more user-friendly: each question prompted to the user is pronounced by a synthetic voice.

The server uses Festival [11] speech synthesis system to generate speech corresponding to the input text in real-time. The Festival system is free, and can operate as a general multi-lingual speech synthesis system. Currently, the TTS engine integrated on the server supports speech synthesis in English, French, and German languages, but only the English one is currently used.

There is a list of different synthesized voices to choose from. The list is displayed with a drop-down button in the welcome screen of the game. The player can try out and choose any voice at the beginning of a game. Other languages could be added to the server with new language packages.

The game starts with a welcome message from the game to the player, generated on the TTS server. The client, at every stage of the game, makes a request to the TTS server with the required text to be synthesized. The text can be a welcome message, a concluding message, or a question and corresponding options, depending on the current state of the game. The TTS server, upon receiving the text, generates an audio file corresponding to the text using the voice selected by the player. Further, this audio file is transmitted back to the client via an HTTP response. Then, the client plays this audio file back to the player.

4.3 ASR Server

The third and most important module of the server is the Automatic Speech Recognition (ASR) engine. In order to improve the performance compared to the earlier version of the game, a Kaldi-based online speech recognizer is exploited. More particularly, a Kaldi-based keyword spotting system is implemented instead of a full-fledged speech recognizer. The reasons for this are two-fold:

- First, at each stage of the game, there are only a limited set of options (closed dictionary phrases) to be recognized. This also allows to return a confidence about the detected key-word, by normalizing the key-word with respect to a universal backgorund model (UBM).
- Second, the performance of a keyword spotting system can be much higher than a conventional ASR.

The keyword spotting system is implemented by switching the language model to apply at each stage of the game. These language models are compatible with Kaldi, and are capable of matching a finite set of keywords.

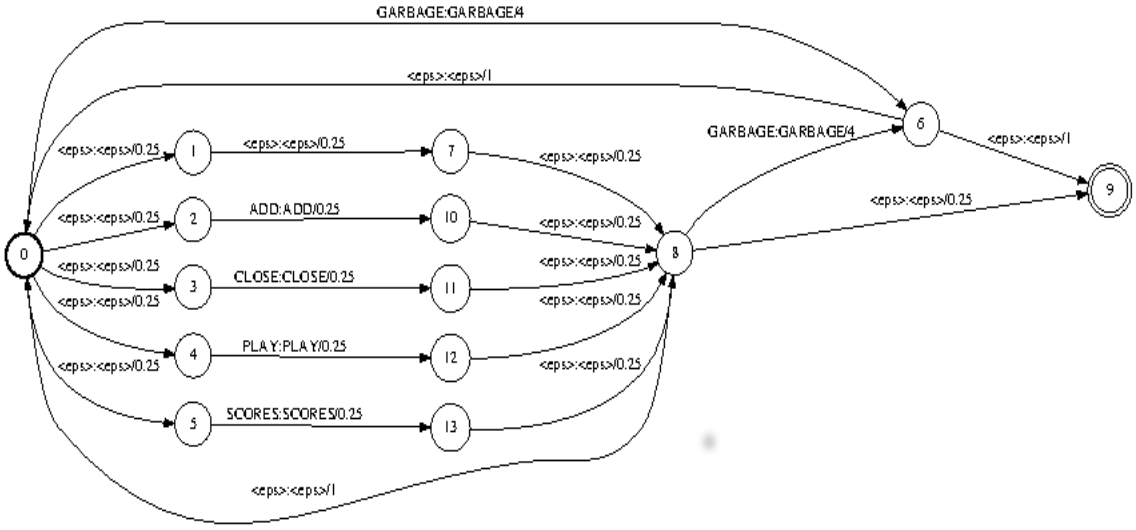

Figure 4: An Example Graph for a Keyword Spotting System

A script was written to generate these language models automatically from a list of keywords. The most specific part of the script is an "awk" script that generates a word-based FST in a text format.

As an input, the script is expected to be provided with a list of phrases, and will provide as output a language model "graph" that can be used by the kaldi speech recogniser. Note that partial matches of the phrases will also be allowed.

All the phrases in the list will have the same weight; that weight is normally spread equally among the words of the phrase. Identical subphrases are grouped together in the FST. Individual words of the phrases are separated by optional silences.

A "garbage/UBM" model with a much higher weight (i.e. a lower probability to be selected) is then added to the FST. In this model, all phones are equi-probable and it can loop on itself, go back to the start node, or can appear after any phrase. This enables the garbage model to represent any other phrase which is not present in the graph. An example graph for 4 different keywords is shown in Figure 4.

Now, from the FST representation, the general scripts to generate graphs in Kaldi are exploited. This FST will be used to generate the "grammar" part of the decoding FST graph. On top of the grammar, a lexicon is automatically generated from the list of unique words using phonetisaurus [12], with models trained on CMUdict [13]. Up to 3 variants of the word pronunciations are generated each time, along with their probabilities.

Finally, the graph is linked to the normal acoustic models used for English online decoding in DBOX. These acoustic models are trained [14] on

icsiami meeting data using the online2 recipes from the Kaldi speech recognition toolkit.

5 Conclusion

This work implements a real-time voice activity detector and modifies the gaming interface accordingly to make it more convenient for the end-user. The new version integrates a Kaldi-based keyword spotting system to enhance the performance. In order to make the KWS faster, the language models are generated offline for different questions and is used during the recognition process accordingly. Thus, the work successfully implements different real-time speech technologies for online gaming. The game currently supports only one language, English. Further work can be done on making the game multilingual (e.g. include French, German etc.). For this, the game interface should include an option to select the language by the user. The database for questions-answers is already in multilingual format, and the TTS server also supports several languages. But the keyword spotting system needs acoustic and language models from other languages; and some changes will be required in the recognizer to switch between recognizers corresponding to different languages.

References

[1] H. Gasimov, A. Triastcyn, P. Motlicek, and H. Bourlard, "Who wants to be a millionaire?" Idiap, Rue Marconi 19, Idiap-Com Idiap-Com-03-2012, 7 2012.

[2] H. Gasimov, P. Motlicek, and H. Bourlard, "Who wants to be a millionaire? (ii)," Idiap, Rue Marocni 19, Martigny, Switzerland, Idiap-Com Idiap-Com-02-2013, 2 2013.

[3] S. Young, G. Evermann, M. Gales, T. Hain, D. Kershaw, X. Liu, G. Moore, J. Odell, D. Ollason, D. Povey *et al.*, *The HTK book*. Entropic Cambridge Research Laboratory Cambridge, 1997, vol. 2.

[4] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, "The kaldi speech recognition toolkit," in *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, Dec. 2011, iEEE Catalog No.: CFP11SRW-USB.

[5] M. Moattar and M. Homayounpour, "A simple but efficient real-time voice activity detection algorithm," *EUSIPCO. EURASIP*, pp. 2549–2553, 2009.

[6] Dbox project. [Online]. Available: http://www.idiap.ch/project/d-box/

[7] Shout. [Online]. Available: http://shout-toolkit.sourceforge.net/index.html

[8] Apache tomcat. [Online]. Available: https://tomcat.apache.org/tomcat-8.0-doc/

[9] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee, "Hypertext transfer protocol–http/1.1," 1999.

[10] Hsqldb. [Online]. Available: http://hsqldb.org/web/hsqlDocsFrame.html

[11] Festival tts. [Online]. Available: http://www.cstr.ed.ac.uk/projects/festival/

[12] J. R. Novak, "Phonetisaurus: A wfst-driven phoneticizer," http://code.google.com/p/phonetisaurus/.

[13] ArpaBet, "CMU pronouncing dictionary," http://www.speech.cs.cmu.edu/cgi-bin/cmudict, Feb 2013.

[14] P. Motlicek, D. Povey, and M. Karafiát, "Feature and score level combination of subspace gaussinas in lvcsr task," in *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on.* IEEE, 2013, pp. 7604–7608.