OPEN AND EXTENDABLE SPEECH RECOGNITION APPLICATION ARCHITECTURE FOR MOBILE ENVIRONMENTS

Tanel Alumäe[*]

Institute of Cybernetics
Tallinn University of Technology, Estonia

Kaarel Kaljurand

Institute of Computational Linguistics
University of Zurich, Switzerland

ABSTRACT

This paper describes a cloud-based speech recognition architecture primarily intended for mobile environments. The system consists of a speech recognition server and a client for the Android mobile operating system. The system enables to implement Android's Voice Input functionality for languages that are not supported by the default Google implementation. The architecture supports both large vocabulary speech recognition as well as grammar-based recognition, where grammars can be implemented in JSGF or Grammatical Framework. The system is open source and easily extendable. We used the architecture to implement Estonian speech recognition for Android.

Index Terms— Speech recognition, CMU Sphinx, mobile devices, Android, open source, Estonian, Grammatical Framework

1. INTRODUCTION

Entering text on a small mobile device is often sluggish and error-prone, compared to text input via full-sized keyboard. This fact has been noticed by mobile software manufacturers and today many phone operating systems offer text input by means of automatic speech recognition (ASR). For example, on the Android platform, Google Voice Search (application that lets user search the web using a spoken query) [1] is supported in 29 languages and accents (at the end of January 2012).

Although the breath and width of Google's language technology support is remarkable, there are many reasons not to rely only on the official vendor for having ASR on mobile devices. First, there are still numerous languages with a large number of speakers for which platform-native ASR probably won't be supported anytime in the future. Second, there are reasons one might want to replace the built-in ASR service with a custom implementation also for already supported languages: an alternative service might provide better control over privacy, it may support more features and it may be useful for scientific reasons.

[*]This work was partly funded by the Estonian Ministry of Education and Research target-financed research theme no. 0140007s12.

Estonian is the official language of Estonia, spoken by about one million people. It is a Uralic language and is closely related to Finnish. The need for support for language technology has been recognized by the Estonian government and the field is now actively supported [2]. In recent years, several speech recognition system prototypes for Estonian have been developed [3, 4]. However, the support for Estonian in commercial products, especially concerning spoken language, is still lacking.

This paper describes a cloud-based ASR system mainly intended for mobile environments. We give an overview of the server's features, the client applications for Android and describe public deployment of the system for the Estonian language. The developed software is open source, easily extendable and adaptable for other languages.

2. DESIGN AND IMPLEMENTATION

2.1. Requirements

As many aspects of our system were modeled after Google Voice Search, our goal was to provide a similar user experience for Estonian as Google Voice Search offers for its supported languages. Thus, the main design goals for the usability of our system were:

- ASR accuracy: although the system is a research prototype, we still have to provide acceptable word error rate for people to use it.
- Low latency, where latency is defined as the time from when the user finishes speaking until the result appears on the screen.

In addition to usability, we had a number of aspects in mind when designing the architecture of the system:

- Scalability: it must be easy to scale the system to a large number of concurrent users.
- Language independence: it must be possible to port the system to other languages with minimal effort.
- Extendability and flexibility: the system must support both free-form large vocabulary continuous speech

recognition (LVCSR) as well as grammar-based speech recognition with limited vocabulary. Furthermore, it should be easy to add new features to the system, e.g., support for different kinds of grammars, support for adaptation based on different aspects of the query, support for various audio encodings, etc.

- Free and open source: we wanted the system to be completely free, so that it could be modified and deployed with minimal cost.

2.2. Implementation

As with most ASR systems for mobile devices, our system is also based on cloud technology: speech is recorded on the device and sent over mobile data connection to a server which delivers the recognition result. While this approach has some disadvantages (such as requirement for a data connection, privacy issues), it also has a number of advantages, which have been discussed in detail in previous papers [1].

Our ASR server is based on various open-source technologies. The core of the system is the Pocketsphinx decoder of the CMU Sphinx speech recognition toolkit [1] . We selected this decoder as opposed to many other freely available recognition engines because we have found it to be accurate, fast and have a relatively small memory footprint. The decoder is integrated with other parts of our server via its GStreamer interface. We also use the GStreamer framework [2] for decoding and resampling audio from various formats. The main request handling and management code is written in the Ruby programming language and uses the Sinatra web framework [3] .

The ASR server has a modular architecture, with different modules responsible for trigram and grammar-based recognition. Clients specify the language model when making the recognition request. Clients can also upload new grammars in JSGF and Grammatical Framework formats. Source code of the server is available with the BSD-license [4] .

2.3. Support for Grammatical Framework grammars

Most common grammar formats used in speech recognition (JSGF, SRGS, GSL, grXML) are based on (augmented) Backus-Naur Form. Grammatical Framework (GF) [5], on the other hand, is a functional programming language for grammar engineering. The grammar author implements an abstract syntax and its corresponding concrete syntax and by doing that describes a mapping between language strings and their corresponding abstract syntax trees. This mapping is bidirectional — strings can be parsed to trees, and trees linearized to strings. This architecture supports multilinguality as there can exist multiple concrete syntaxes corresponding to a single abstract syntax, and the respective languages can be thus automatically translated from one to the other.

GF is optimized to handle natural language features like morphological variation, agreement, long-distance dependencies, etc. As a grammar formalism GF covers context free languages and even goes beyond. GF grammars can be implemented in a modular fashion and tested e.g. by random or exhaustive generation of abstract trees and their linearizations. GF comes with various tools that cover grammar authoring, compatibility with most popular programming languages, conversion into other grammar formats (including several speech recognition formats), and a reusable grammar library for $\sim$20 world (natural) languages.

2.4. Web service API

Communication with the recognition server is loosely based on the (undocumented) Google Speech API used by the Chrome browser to implement the W3C Speech Input API.

To invoke a recognition request, client makes a HTTP POST request to the web service URL, with speech data in the request body. Currently, the server supports raw, wav, ogg, MPEG and FLAC audio encodings. Server responds with a JSON object that contains the results. Multiple N-best hypotheses are optionally returned, as shown below.

```
{
  "status": 0,
  "hypotheses": [
    {"utterance": "see on esimene lause"},
    {"utterance": "see on esimene lause on"},
    {"utterance": "see on esimene lausa"},
  ],
  "id": "61c78c7271026153b83f39a514dc0c41"
}
```

By default, an n-gram language model configured in the server is used to recognize speech. To use grammar-based recognition, the grammar file has to be first uploaded to the web service. Then, client can specify the grammar identifier in the recognition query to request recognition based on the grammar.

As explained earlier, GF grammars can contain concrete grammars in several languages. When making a recognition query using a GF grammar, clients can specify the input and output language of the GF grammar: input language grammar is used for recognition, and the recognized utterance is automatically translated to the output language(s). The example below using a toy robot grammar demonstrates this: the utterance '*mine neli meetrit edasi*' ('go four meters forward') is translated to English and to an "application" language (presumably used by the robot), the result is given in the field `linearizations`:

```
{
  "status": 0,
  "hypotheses": [
```

[1] [http://cmusphinx.org]
[2] [http://gstreamer.freedesktop.org]
[3] [http://www.sinatrarb.com]
[4] [http://github.com/alumae/ruby-pocketsphinx-server]

```
    {
      "utterance": "mine neli meetrit edasi",
      "linearizations": [
        { "lang": "App",
          "output": "4 m >" }
        { "lang": "Eng",
          "output": "go four meters forward" },
      ]
    }
  ],
  "id": "e2f3067d69ea22c75dc4b0073f23ff38"
}
```


The web service supports HTTP chunked transfer encoding for receiving request data. This enables a client to start sending speech data to the server instantly, before the user has finished speaking (and before the length of the request is known). As a result, the server can start decoding the stream before receiving the end of the request. Although this makes recognition more error-prone, as cepstral mean normalization (CMN) cannot be performed before decoding, it dramatically decreases recognition latency. To reduce the problem caused by CMN, we cache the CMN coefficients from a client between individual requests, and use the saved coefficients for new requests, updating the coefficients as new data arrives.

3. SPEECH RECOGNITION MODELS FOR ESTONIAN

The acoustic and language models that the Estonian speech recognition service uses are largely based on previously developed research systems [3, 4]. Acoustic models were trained on various wideband Estonian speech corpora: the BABEL database of dictated speech, a corpus of Estonian broadcast news, a corpus of broadcast conversations, a corpus of lecture and conference recordings, and a corpus of spontaneous dialogues, totalling in around 90 hours. The models are triphone HMMs, using MLLT/LDA-transformed MFCC features, with 3000 tied states, each modelled with 32 Gaussians. The model inventory consists of 25 phonemes and 7 silence/filler models. Cepstral mean normalization was applied.

Language model (LM) for large vocabulary recognition was trained on various text corpora: newspapers, magazines, parliament transcripts, fiction, scraped web data (blogs, news portals), Wikipedia, broadcast news transcripts, transcripts of broadcast conversations, lecture transcripts (250M words in total). Separate models were built from each source and linearly interpolated into the final model. As we didn't have any development data before the launch, the interpolation weights for the initial model were optimized on the development data from a broadcast conversations transcription task [4]. As Estonian is a heavily inflected and compounding language, we split compound words into separate LM units, and reconstruct compound words from recognition hypotheses using a conditional random field model. The LM vocabulary consists of most likely 200K units. Due to the limitation of Pocketsphinx, trigram LM was used. Estonian is an almost phonetic language which lets us use a simple hand-built transducer for building a pronunciation dictionary, with a list of exceptions for common foreign names.

4. APPLICATIONS

4.1. Overview

We developed two Android applications that use the Estonian ASR server: *Kõnele*, a generic large vocabulary Estonian voice recognition service, and *Arvutaja*, a grammar-based application for performing some useful tasks by voice.

Android provides interfaces and infrastructure (*RecognizerIntent* and *RecognitionService* classes) that allow third-party implementations transparently offer speech recognition service to other applications. By default, this service is handled by the built-in Google service. Our application *Kõnele* [5] (*'kõnele*' is the imperative form of 'to speak' in Estonian) implements this service for Estonian voice input. Android applications which allow speech input typically have a small microphone button as part of their user interface (typically next to a text field). Such applications include keyboard apps, applications with a search bar (Google Maps, YouTube, etc.), and otherwise a wide variety of applications for which speech input makes sense or is simply the only way for entering complex information (e.g. quick notes and hands-free navigation applications, Siri-like intelligent assistant applications). By implementing the necessary interfaces, Estonian speech recognition can be used in all such applications without modification.

One additional feature of the *Kõnele* application is the possibility to assign different grammars to different applications, via its configuration panel. If the *Kõnele* service is called by an application for which there exists a grammar assignment, then the service automatically requests the ASR server to use the configured grammar for speech recognition, instead of the generic n-gram model. This can have two uses: first, by using an application-specific grammar, speech recognition can be made more accurate (e.g., a navigation application can be assigned a grammar that accepts addresses of a city or a small country). Second, by using a GF grammar, one could use speech in Estonian, and have the result automatically translated to application language (e.g., English), this allows e.g. performing computations in WolframAlpha even though its query language is English-based, or setting alarms and timers via existing Android intelligent assistant apps, even though these expect English or symbolic/numerical input.

[5] [https://play.android.com/details?id=ee.ioc.phon.android.speak, source: http://github.com/Kaljurand/K6nele]

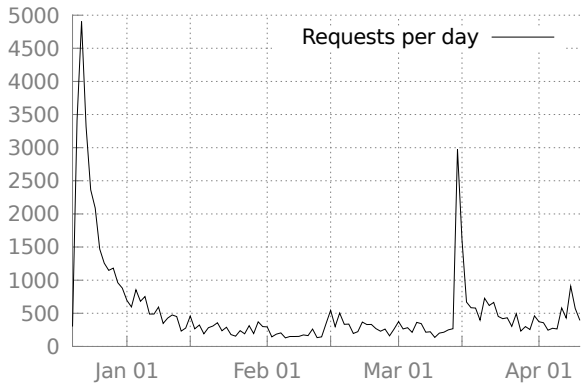

Fig. 1. Number of recognition requests per day to the ASR server by Android applications. Peaks in the graph correspond to mentions of the mobile apps in popular mass media.

The application *Arvutaja* (literally 'the one who calculates')[6] was implemented as an example of grammar-based speech recognition application. The application allows performing the following tasks by Estonian speech: evaluating arithmetical expressions (e.g., 'fifty two point two times six', converting units and currencies (e.g. 'one hundred euros in dollars'), performing Estonian address queries (e.g., 'Valgevase five Tallinn'). Results of calculations are computed by the application itself, address queries are directed to Google Maps (or any other application that handles addresses).

4.2. Deployment

The applications were published in Android Market at the end of December 2011. After their demonstration in national television, they gained popularity, and the generic voice recognition service *Kõnele* was the most downloaded free Android application in Estonia for almost a week. Since then, the number of daily downloads has been decreasing. The install base of the applications is around 5000 in total. Figure 1 shows the number of recognition requests made through the Android applications to the ASR server since the launch.

4.3. Accuracy

To estimate the accuracy of the ASR service, we transcribed a random selection of utterances that were sent to our server for recognition. As we were mainly concerned of the quality of n-gram based recognition, we only picked queries that didn't request to use any grammar (i.e., we excluded the requests made by *Arvutaja*). The average word error rate (WER) turned out to be 44.9%. Then, we took all requests from our logs (excluding the test set), and applied unsupervised acoustic model adaptation: the requests were transcribed by a more complex three-pass recognizer (using fourgram LM, with speaker-dependent CMLLR and MLLR adaptation between passes), and estimated speaker-independent CMLLR and MLLR transforms for the initial acoustic models. This had a remarkable impact on the recognition quality – WER dropped to 33.4%. We also reoptimized the interpolation weights of the trigram LM based on automatically transcribed data, which reduced the WER further to 28.4%.

[6][https://play.android.com/details?id=ee.ioc.phon.android.arvutaja, source: http://github.com/Kaljurand/Arvutaja]

5. CONCLUSION

We implemented an open and extendable ASR architecture that enables to develop speech-based applications for mobile platforms. The system stack consists of an ASR server and a client service for Android that transparently provides LVCSR functionality for other applications. The service also supports grammar-based ASR, and we used this feature to implement a small assistant-like application. The applications were publicly launched at the end of 2011, gaining quickly over 5000 installations for a target population of less than one million speakers. We plan to extend the system with features targeted towards speech collection, that would enable acoustic model adaptation for individual speakers and collection of speech corpora.

The framework makes it relatively easy to develop mobile speech input applications for other languages that are, for example, not yet supported natively by the OS vendor. The support for grammar based decoding allows to create limited-vocabulary applications even for languages that don't have enough training data for creating robust LVCSR models.

6. REFERENCES

[1] Johan Schalkwyk, Doug Beeferman, Françoise Beaufays, Bill Byrne, Ciprian Chelba, Mike Cohen, Maryam Kamvar, and Brian Strope, ""Your Word is my Command": Google search by voice: A case study," in *Advances in Speech Recognition*, Amy Neustein, Ed., pp. 61–90. Springer US, 2010.

[2] Einar Meister and Jaak Vilo, "Strengthening the Estonian language technology," in *LREC 2008*, Marrakech, Morocco, 2008.

[3] Tanel Alumäe and Einar Meister, "Estonian large vocabulary speech recognition system for radiology," in *Baltic HLT 2010*, Riga, Latvia, 2010, pp. 33–38.

[4] Tanel Alumäe and Ahti Kitsik, "TSAB – web interface for transcribed speech collections," in *Interspeech 2011*, Florence, Italy, 2011, pp. 3335–3336.

[5] Aarne Ranta, *Grammatical Framework: Programming with Multilingual Grammars*, CSLI Publications, Stanford, 2011.