Controlled natural language in speech recognition based user interfaces

Kaarel Kaljurand[1] and Tanel Alumäe[2]

[1] Institute of Computational Linguistics, University of Zurich, Switzerland,
kaljurand@gmail.com
[2] Institute of Cybernetics, Tallinn University of Technology, Estonia,
tanel.alumae@phon.ioc.ee

Abstract. In this paper we discuss how controlled natural language can be used in speech recognition based user interfaces. We have implemented a set of Estonian speech recognition grammars, a speech recognition server with support for grammar-based speech recognition, an Android app that mediates the communication between end-user Android apps and the speech recognition server, and an end-user Android app that lets the user execute various commands and queries via Estonian speech. The overall architecture is open and modular, offers high precision speech recognition, and greatly simplifies the building of mobile apps with a speech-based user interface. Although our system and resources were developed with the Estonian speaker in mind and currently target a small number of domains, our results are largely language and domain independent.

Keywords: grammar-based speech recognition, user interfaces, controlled natural language, Grammatical Framework, Estonian

1 Introduction

Speech recognition based user interfaces can be effective in many environments. The only requirements are the lack of major background noise and privacy/security concerns that one might have when audibly communicating with the machine. In many domains such user interfaces provide the most efficient form of human-machine communication because alternative interfaces (e.g. keyboard, visual menu system, etc.) are not available or are cumbersome to use. For example, when driving a car, one's eyes and hands are occupied with driving while the speech faculty is freely available. Recently a general application platform for speech based user interfaces has emerged in the form of mobile devices (smartphones and tablets). Such devices feature a small display and a small (and often only virtual) keyboard. Many types of otherwise simple tasks (looking up a phone number, launching an application, setting an alarm, etc.) can become time consuming and cumbersome if performed on a mobile device. Performing such tasks via speech can solve this problem, and this has already been demonstrated by applications like Apple's Siri and Google's Voice Actions, which have become quite popular among users [6].

A controlled natural language (CNL) for speech recognition is a CNL like any other — it has a precisely defined syntax, its sentences have a formal (executable) meaning, and it comes with an end-user documentation describing its syntax, semantics and usage

patterns. A CNL for spoken input differs from a CNL for written input because its design has to take into consideration the properties of speech: there are words which sound the same, or can sound the same if distorted by background noise, some character sequences cannot be (directly/reliably) pronounced (numbers, punctuation), utterances are syntactically simpler, etc.

Many applications naturally feature grammatically and vocabulary-wise reduced "language", examples include a calculator, a measurement unit converter, a car navigation system, an address book browser, an alarm clock, a flight booking website's form filling interface. Basing the interaction with such applications on a CNL offers two benefits — (1) the recognition of user input works in a precise way, (2) it becomes much easier for the application developer to map the user input to the application's executable code (e.g. the formal expression that actually sets the time on the alarm clock).

In this paper we describe a set of Estonian speech recognition grammars that cover the language of some important tasks that one normally performs on a mobile device. The grammars have been implemented in Grammatical Framework (GF) [9] making them easily extendable and portable to other languages and application domains. We also describe the overall architecture of our speech recognition system which combines an online speech recognition server, two Android-based apps and a repository of grammars. This modular architecture simplifies the building of natural and easy to use interfaces to applications, potentially ranging from a simple alarm clock to in-car navigation systems and control panels for the "smart house".

In section 2 we review related work; in section 3 we discuss the special features demanded by speech recognition oriented CNLs; in section 4 we briefly introduce GF; in section 5 we provide an overview of the grammars that we have developed; in section 6 we describe the architecture of the overall speech recognition system; in section 7 we summarize our main results; and in section 8 we mention some loose ends and general future work.

2 Related work

Speech applications are a well-studied field which also includes grammar-based and even GF-based applications. The use of GF in speech applications has been studied in [4] and resulted in multi-modal systems which combine speech input with input from the touch screen. These systems can also hold a dialog where the eventual result is obtained after a sequence of conversation turns. In addition these systems are multilingual allowing input/output in several natural languages. The concrete applications include ordering a pizza (with various toppings and drinks) and asking for directions in a city public transportation system. GF is used as a framework for describing the semantic model of the system and its various expressions in natural and formal languages (e.g. image of the pizza with the specified toppings). Compared to this work, our current system is simpler in several dimensions (single language, single modality, no dialog), we have rather focused on developing a general platform on which more sophisticated applications can be easily built.

An architecture similar to ours is developed in [13]. It consists of a mobile application for calendar events management by English speech, a Nuance[3] speech recognition server, and a Regulus grammar [11]. In addition to grammar-based speech recognition, the system also uses a statistical recognizer that is applied to out-of-grammar input with an eventual goal of guiding the user towards supported phrases. The built-in help system makes the architecture more sophisticated than ours, but the overall system is much harder to deploy as it relies on commercial closed-source software (Nuance, SICStus Prolog). We did not explore if the system would be portable to Estonian and to a free/open platform.

In general, the possibility of speech input is available on all the major smartphone platforms (Android, iOS, Windows Phone) via end-user applications such as Google Voice Actions and Siri. However these applications do not support user-specified speech recognition grammars and offer only a limited API and configurability. They are also closed source and thus not portable to other domains and languages by third parties.

Computational approaches to the Estonian language have so far focused on large coverage shallow parsing, detailed description of morphology and word senses (WordNet), statistical methods for machine translation and speech recognition, etc. [7]. For the most part, the developed resources are not directly reusable for building controlled Estonian grammars. The existing tools for morphological synthesis could be used in principle to automatically generate wordforms for the grammars, but for the current system we decided not to integrate these tools and generate the required wordforms with *ad hoc* rules formulated using GF's regular expressions.

3 Requirements

Spoken language differs from written language in various ways which must be reflected in the design of speech recognition oriented CNLs. Certain written forms and orthographic conventions are not reflected in speech. For example:

- homophones (e.g. 'cite', 'site', 'sight') cannot be distinguished with only acoustic cues (in Estonian, homophones occur rarely, e.g. in the case of word-initial plosives, e.g. 'baar' (*bar*) and 'paar' (*pair*) are pronounced in the same way);
- there is no necessary break between words as there is in written speech, creating confusion pairs ("oronyms"), such as 'ice cream' vs. 'I scream', 'depend' vs. 'deep end';
- numbers, abbreviations, URLs are expanded to words (e.g. '3G' $\mapsto$ 'three gee'); often, the expansion is non-deterministic (e.g. '1990' can be verbalized as 'nineteen ninety', 'one thousand nine hundred and ninety' or 'nineteen hundred and ninety');
- punctuation symbols are typically not verbalized.

CNLs in speech recognition applications should avoid short or unpronounceable words, and avoid similarly sounding words that occur in the same syntactic position. These CNLs cannot rely on technologies like look-ahead editing [12] to guide the user towards the completion of syntactically correct sentences. The sentences must therefore

[3] `http://nuance.com/`

be shorter and syntactically simpler. Also, because many existing speech recognition engines rely on finite-state automata technology, the CNL might have to be expressible by a regular grammar.

When choosing the formalism to implement our grammars we found the following requirements important:

- support for a declarative description of how the raw speech transcription (e.g. 'half past six in the evening') maps to the formal application language ('18:30');
- built-in handling of the complexities of natural language, e.g. the Estonian morphology;
- user-friendly syntax and semantics and/or editing environments which allow also novice grammar engineers to write well-functioning grammars;
- compatibility with open source speech recognition toolkits;
- support for standard software engineering practices (reusable modules, unit and regression testing, etc.).

Existing standard speech recognition grammar formalisms (such as SRGS[4] and JSGF[5]) do not cover all these requirements. They are usually simple BNF languages without any special support for natural languages nor the capability to assign a formal meaning to the language described by the grammar.

4 Grammatical Framework

In order to implement our CNLs we chose Grammatical Framework (GF) [9] because it covers our requirements and has been successfully used before to build speech applications [4]. GF 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. As this mapping is bidirectional — strings can be *parsed* to trees and trees *linearized* to strings — this architecture supports multilinguality. There can exist multiple concrete syntaxes corresponding to a single abstract syntax, and the respective languages can be automatically *translated* from one to the other.

GF is optimized to handle natural language features like morphological variation, agreement, long-distance dependencies, etc. In terms of the expressivity of the grammar formalism GF covers context free languages and even goes beyond. GF grammars can be implemented in a modular fashion and tested by random or exhaustive generation of abstract trees and their linearizations. GF comes with various tools that cover grammar authoring, compatibility with many popular programming languages, conversion into other grammar formats (incl. several speech recognition formats), and a reusable grammar library for $\sim$25 natural languages (unfortunately excluding Estonian at the time of writing).

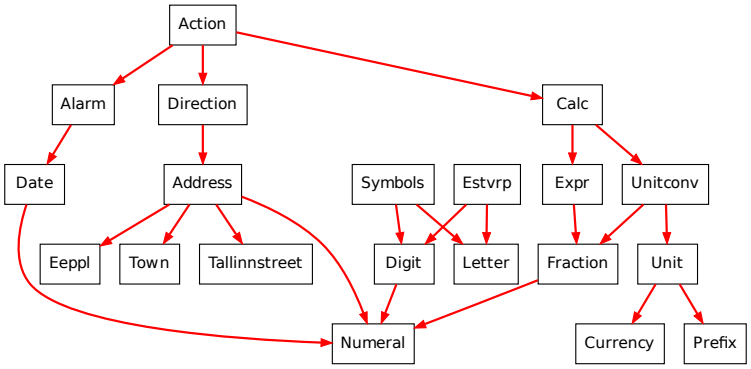

Fig. 1. The hierarchy of abstract grammar modules. The largest grammar is *Action*, which covers arithmetical, unit conversion, alarm setting and address query expressions. The most imported grammar is *Numeral* which is a building block in all the mentioned expressions.

5 Grammars

We have developed several grammars targeting the general topics of time, location and numerical calculation. A GF grammar is a set of concrete language definitions with their corresponding single abstract language definition. In our more specialized sense each grammar contains at least two types of concrete syntaxes, which we refer to as *parsing syntax* and *linearization syntax*. The parsing syntax

- is used for parsing;
- directly corresponds to natural language speech, e.g. it uses words (‘two’, ‘plus’), not symbols (‘2’, ‘+’);
- allows ambiguous input (i.e. produces multiple parse trees);
- allows variants (synonyms).

The linearization syntax on the other hand

- is used for generation;
- typically corresponds to a machine language (where symbols are not necessarily pronounceable);
- does not require the presence of syntactic sugar.

The meaning of the speech input is assigned by parsing it with the parsing syntax and then linearizing the obtained abstract tree with the linearization syntax. A grammar can be easily extended by adding a new concrete language that provides the linearization

[4] `http://www.w3.org/TR/speech-grammar/`
[5] `http://www.w3.org/TR/jsgf/`

of all the functions already described in the abstract syntax. The new concrete language can either stand for a parsing syntax (i.e. support for a new natural language) or the linearization syntax (i.e. support for a new machine format).

In our case, the parsing syntax of our grammars corresponds to Estonian speech and the linerization syntax targets the input language of an existing application such as standard calculator, Google Maps, or WolframAlpha. Note however that in most cases the actual language supported by these tools is not publicly and precisely documented, nor can it be considered a stable API. We next provide a more detailed look into the main grammars.

GF offers various support for modularity, including an extension relation between grammars, which we use to share the implementation of numbers and to build union grammars (see figure 1).

5.1 Direction

The *Direction* grammar describes Estonian places by containing a list of settlement names, a list of street names, positive integers to stand for house numbers and a phrase for combining two place names in a directions query. As there are many place names this grammar contains a large number of terminals, but is syntactically very simple, covering the patterns

```
Placename = Streetname Housenumber (Town) (Country)
Placename = Town (Country)
Direction = From Placename To Placename
```

where the optional `Town` and `Country` can be mapped to some reasonable defaults (e.g. 'Tallinn' and 'Estonia') if they are missing. All the place names are in the nominative case.

The linearization syntax targets the language understood by Google Maps. For the most part it is identical to the parsing syntax, i.e. it contains the same terminals. The only difference are the *from* and *to* phrases. The following examples list the input utterance in Estonian, its corresponding machine format, and its English translation (for the purposes of this paper).

Example 1.
algus akadeemia tee kaks kümmend üks lõpp räpina
`FROM Akadeemia tee 21, Tallinn TO Räpina, Estonia`
begin Akadeemia street 21 end Räpina

The *Direction* grammar is compiled on the basis of two freely available lists of Estonian place names:

- settlements in Estonia (towns, villages, etc.), 4300 names from GeoNames[6];
- names of streets in Tallinn, 1500 names from the place names' resource of the Institute of the Estonian Language[7].

[6] `http://www.geonames.org/`
[7] `http://www.eki.ee/knab/`

At the moment the grammar does not model naming variation and ambiguity, i.e. bijection holds between the set of places (abstract functions) and the set of place names (their linearizations). This has the consequence that disambiguation must be performed by an external application (e.g. Google Maps).

The grammar currently lacks the street names of Estonian towns other than Tallinn, Estonian names of foreign places (e.g. 'Venemaa', 'Riia'), and names of places which are not towns nor buildings with a street number (e.g. parks, landmarks). It should be noted that the grammar does not include a statistical component that would e.g. assign a lower prior probability to smaller places which are rarely visited (e.g. a short street with just one house). Also it allows house numbers up to 999 with any street name, i.e. this grammar cannot be used in applications that use exhaustive generation or look-ahead editing because it can generate addresses which do not exist.

5.2 Expr

The arithmetical expression grammar *Expr* describes positive integers up to 10^{12} (following the abstract syntax of the GF numerals grammar [5]), their negative counterparts and their combinations with a dot (to cover some of the rational numbers). The numbers can be combined with the standard arithmetical operators of addition, subtraction, multiplication, division, and exponentiation in order to form (possibly infinitely long) sentences. Our implementation follows the example provided in [9], but only the left-associative interpretation is supported and all operators are considered to have equal precedence.

Example 2.
kaks pluss kolm miinus miinus neli korda viis jagatud kuus astmel seitse koma sada
`((((2 + 3) - (-4)) * 5) / 6) ^ 7.100)`
two plus three minus minus four times five divided-by six to-the-power-of seven point hundred

The strings that correspond to the formal expressions can be directly evaluated with any standard calculator, e.g. the one included in Google Search.

5.3 Unitconv

The unit conversion grammar *Unitconv* includes the same numbers as the *Expr* grammar, ∼50 base measurement units (of ∼10 physical quantities), and ∼15 world currencies. The grammar describes how the base units form more complex units via

- SI prefixing: 'meeter' (`m`) → 'kilo meeter' (`km`),
- exponentiation: 'senti meeter' (`cm`) → 'kuup senti meeter' (`cm^3`), and
- fractions: 'kilo meeter' (`km`) and 'tund' (`h`) → 'kilo meetrit tunnis' (`km/h`).

All these constructors take into account the types of the input components (e.g. length) and can thus generate the correct complex unit (e.g. volume). A number and two units can be combined into a unit conversion expression as shown in the examples.

Example 3.
viis koma kaks meetrit jalgades
`convert 5.2 m to ft`
five point two meters in feet

Example 4.
viis miili ruut tunnis meetrites ruut sekundis
`convert 5 mi*h^-2 to m*s^-2`
five miles per square hour in meters per square second

Example 5.
viis ameerika dollarit rootsi rahas
`convert 5 USD to SEK`
five American dollars in Swedish money

The grammar supports some variation and ambiguity, e.g. 'kroon' (*crown*) refers to multiple currencies (SEK, NOK, DKK, ...), unless disambiguated by e.g. 'rootsi kroon' (*Swedish crown*) or (if the user does not know the name of the currency) 'rootsi raha' (*Swedish money*).

The nouns corresponding to units can potentially have 3 different inflectional endings depending on their role in the sentence. The morphological forms are calculated automatically from the partitive base form, making the extending of the grammar with new units almost as simple as extending the *Direction* grammar with new place names.

The linearization syntax targets the unit conversion syntax supported by Google Search and WolframAlpha, i.e. it consists of English phrases ('convert') and ASCII conventions for writing formulas (`m*s^-2`).

5.4 Alarm

The *Alarm* grammar targets one of the most widely performed tasks on mobile devices — setting alarms. The grammar supports pointing to an exact minute in a 24h clock (when the alarm should go off) and specifying a duration by a positive integer number of minutes (after which the alarm should go off).

Example 6.
ärata mind kell seitse null üks
`alarm 07:01`
wake me at seven oh one

Example 7.
ärata mind kaheksateist minutit hiljem
`alarm in 18 minutes`
wake me eighteen minutes later

The linearization grammar targets the language understood by some of the "intelligent assistant" apps found on Google Play[8], notably Speaktoit Assistant[9].

[8] `http://play.google.com/store`
[9] `http://www.speaktoit.com/`

5.5 Estvrp and Symbols

The *Estvrp* (Estonian vehicle registration plate) grammar lets one spell the Estonian car registration numbers, which typically consist of three letters followed by three digits (e.g. 'ABC123'). As the speech recognizer discriminates short sounds (e.g. /oo/ vs. /uu/) very unreliably, the grammar denotes letters by a set of selected longer proper names (e.g. 'Artur' for 'A').

There exists also a *Symbols* grammar that describes an arbitrary length sequence composed of letters (*Letter*) and digits (*Digit*).

5.6 Ambiguity

In general, the output of our grammar-based speech recognizer is not a single string but a set of strings. This is because our grammars not only accept/reject the given speech input, but also translate it to another concrete format. For example, the recognized speech might result in a string 'pii' which is further mapped to (= is ambiguous between) three strings 'π', 'Pii village, Estonia', 'Pii street, Tallinn'. One way to deal with such ambiguity is to make sure that the grammar allows for alternative (synonymous) forms which lack the ambiguity, e.g. 'pii küla' (*Pii village*), 'pii tänav' (*Pii street*), 'arv pii' (*the number* π). Support for variation is also generally good because usually there exist many equally probable ways for saying a command or query, even in the case of simple applications like calculators.

Ambiguity can also be exploited in various interesting ways. For example, a grammar that describes currency conversion can decide to include the string 'european currency' and implement it as ambiguous between 'EUR', 'CHF', 'SEK', etc. resulting in multiple translations for the the expression 'convert one dollar to european currency'. The end-user application can either ask the user to clarify which exact currency was meant or alternatively (and more interestingly) visualize the result set as a comparison table or plot.

6 Overall architecture

We next describe the implementation of the client-server architecture of our grammar-based real-time mobile-oriented speech recognition system (see figure 2).

6.1 Speech recognition server

Our speech recognition server [2] offers real-time transcription of short (up to 20 seconds) Estonian speech signals. The server is designed to respond with at most a couple of seconds delay after input speech has stopped independent of the length of the input. The server is thus usable in applications where the transcription must be obtained immediately such as web search or dictation of an SMS message.

The speech recognition server is based on various open-source technologies. The core of the system is the Pocketsphinx decoder of the CMU Sphinx speech recognition

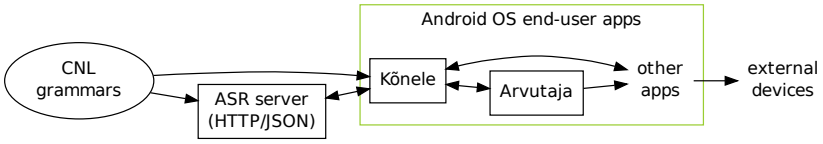

Fig. 2. The overall system consists of an online repository of CNL grammars, an automatic speech recognition (ASR) server that is accessible over HTTP, an Android service *Kõnele* that mediates the communication of end-user apps with the recognition server, and various end-user apps that offer a speech-based UI. Some of such apps (e.g. *Arvutaja*) act as hubs that dispatch incoming speech commands/queries to other apps. Finally, the commands can be carried out on something external to the mobile device (e.g. when opening a garage door by a voice command).

toolkit[10]. We selected this decoder as opposed to 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. The main request handling and management code is written in the Ruby programming language. The source code of the server is available under the BSD-license[11].

The Estonian acoustic models[12] for the speech recognition service were trained on various wideband Estonian speech corpora: the BABEL speech database, a corpus of Estonian broadcast news, a corpus of broadcast conversations, a corpus of lecture and conference recordings, and a corpus of spontaneous dialogs, totaling in around 90 hours. The models are triphone HMMs, using MLLT/LDA-transformed MFCC features, with 3000 tied states, each modeled with 32 Gaussians. The model inventory consists of 25 phonemes and 7 silence/filler models. Cepstral mean normalization was applied.

Speech recognition engines usually rely on statistical (e.g. trigram) language models. Such models are appropriate for many applications of speech technology (dictation of letters, transcription of meetings). By default, our server uses a pre-built trigram model for decoding incoming requests. However, grammar-based decoding can be invoked by specifying the name of the grammar in the request parameters.

Our server allows uploading of grammars in GF's portable runtime format (PGF)[3]. When a new grammar is uploaded, several steps are automatically invoked on the server. First, the concrete syntax of the input language (Estonian) is extracted from the PGF and converted to the JSGF format[13]. The JSGF grammar is further converted to a finite-state automaton that can be used by the speech recognition engine. One of the shortcomings of the current implementation is that the resulting finite-state automaton is a regular approximation of the GF concrete syntax and allows certain inputs that the original GF grammar does not actually allow.

[10] `http://cmusphinx.org`
[11] `http://github.com/alumae/ruby-pocketsphinx-server`
[12] `http://github.com/alumae/et-pocketsphinx-tutorial`
[13] `http://www.w3.org/TR/jsgf/`

The concrete syntax of the input language is assumed to contain orthographically correct words. This allows the server to automatically build a pronunciation dictionary (mapping of words to phoneme sequences) for all the words in the input language. Since Estonian is almost a phonetic language, we can generate pronunciations using a simple transducer, with a list of exceptions for common foreign names [1]. A more flexible architecture would allow the user to provide her own pronunciation dictionary as well as the acoustic models that define the phonemes.

The server has an HTTP interface designed to be similar to the (publicly undocumented) Google's speech recognition server which is used by the Chrome browser to implement the W3C Speech Input API. When a recognition request is made to the server, the name of the PGF grammar can be specified in the request parameters to activate grammar-based recognition. The server uses the regular approximation of the input language concrete syntax of the PGF to decode the audio signal. The resulting recognition hypotheses can be automatically translated to the output language(s) specified in the request. The server is able to return an N-best list of recognition hypotheses for each request. The size of the N-best list can be specified with a request parameter.

The results are returned in the JSON format. The example below shows a transcription of the utterance "mine neli meetrit edasi" ("go four meters forward") and its translation into three concrete languages *Eng*, *Est*, *App*. In this case the hypotheses set contains just a single element.

```
{ "status": 0,
  "hypotheses": [
    { "linearizations": [
        { "lang": "App", "output": "4 m >" },
        { "lang": "Eng", "output": "go four meters forward" },
        { "lang": "Est", "output": "mine neli meetrit edasi" }
      ],
      "utterance": "mine neli meetrit edasi"
    }
  ],
  "id": "d9abdbc2a7669752059ad544d3ba14f7"
}
```

The request fails if the server fails to match the audio signal to the grammar (e.g. because the signal is too noisy or quiet), or if the server recognizes it due to the regular approximation as something which is not covered by the original grammar. Both types of failures are exposed to the user in the same way, and can usually be overcome by repeating the input phrase in a clearer voice.

To our knowledge, the described recognition server is the first free, open source and publicly available web service in the world that supports speech recognition with user-defined grammars.

6.2 Android service *Kõnele*

A wide variety of Android apps support input by speech, typically having a small microphone button as part of their user interface, e.g. keyboard apps, apps with a search bar, apps for in-car use, Siri-like intelligent assistant apps. The Android operating system offers an API[14] (the *RecognizerIntent* and *RecognitionService* classes) that lets such

[14] `http://developer.android.com/reference/android/speech/package-summary.html`

end-user apps call a central service that performs the speech recognition and returns the transcription. In this way the multitude of speech-enabled apps can share the speech recognition provider and do not have to implement this functionality themselves.

We developed an Android app *Kõnele*[15] ('kõnele' is the imperative form of 'to speak' in Estonian) which offers such a speech recognition service to other apps on the device. In addition to being a background service, *Kõnele* also offers a configuration panel which allows the end-users to assign different grammars to different apps by specifying the URL of the grammar and the Java package name of the app. If the *Kõnele* service is called in the context of an app for which there exists a grammar assignment then *Kõnele* sends the grammar URL to the server along with the audio data, which triggers grammar-based speech recognition and translation into the format of the app.

6.3 Android app *Arvutaja*

Arvutaja[16] (Estonian for 'the one who calculates') is a tool that helps the user to evaluate arithmetical and unit conversion expressions, query for directions between Estonian addresses, and set the alarm clock or timer. Commands to *Arvutaja* are given in Estonian speech which is transcribed via *Kõnele*, guided by the union of *Expr*, *Unitconv*, *Direction* and *Alarm* grammars. The transcription is evaluated/executed and displayed to the user by a built-in library (for unit conversion and arithmetical expressions) and/or by an external app (for displaying a route on the map between the given locations or setting the alarm to ring at the given time). In case of ambiguous queries, the user is presented with all the interpretations. Figure 3 shows a screenshot of *Arvutaja*.

The Android framework supports a messaging mechanism called *intents*[17] that lets apps call each other with the purpose of "outsourcing" operations (performing a calculation, displaying a map, setting an alarm). *Arvutaja* uses the intent mechanism to call Google Maps, WolframAlpha, and an alarm clock app for certain tasks. Such an architecture where the user input is directly mapped to an external application to handle this input makes *Arvutaja* defined almost entirely by the externally developed grammars and apps, and thus easily extendable to completely new languages and domains.

6.4 Grammars

An important position in this overall architecture is held by the repository of CNL grammars. This repository makes available the grammars so that they can be accessed both by the speech recognition server and by *Kõnele*. It also facilitates learning to use the grammars and (collaboratively) building new grammars.

Our GF grammars are compiled into the PGF platform independent format and made available via public URLs on GitHub[18], see `http://kaljurand.github.com/Grammars/`. As all the grammars are available in the source format, GitHub can

[15] `http://recognizer-intent.googlecode.com`
[16] `http://kaljurand.github.com/Arvutaja/`
[17] `http://developer.android.com/guide/topics/intents/intents-filters.html`
[18] `http://github.com/`

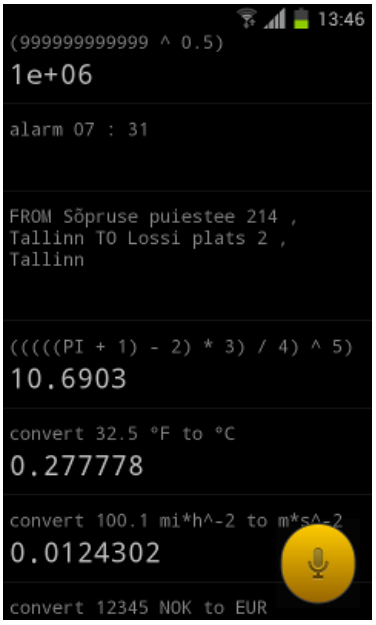

Fig. 3. Screenshot of *Arvutaja*. The user interface prominently includes a microphone button, tapping on which triggers the recording and transcribing of the spoken query. The results are presented in a history list. In some cases the transcription is accompanied by its evaluation. In some cases (e.g. address query), the user needs to tap on the transcription to execute it and view the result (a map) via an external application (Google Maps). Such a user interface is at least on a visual level much simpler than e.g. a typical unit converter interface which usually features a complex menu system for selecting the involved units.

also be used as a collaborative editing environment where the users can easily fork a grammar, extend it and contribute back the modifications. In order to familiarize themselves with the language that the grammars support, the users can look at help documentation and lists of (automatically generated) example sentences. A more sophisticated infrastructure (which is currently not implemented) would also include existing GF tools [10], e.g. Minibar, Translation Quiz, and the online grammar editor[19].

In general, such a repository should make grammar creation easy for everybody. We envision that there can be many different grammar usage scenarios: private grammars used internally in a company, different subset grammars (e.g. address grammars optimized for one country or one town), new grammars with tiny variations with respect to existing grammars (e.g. to account for the variation in an intelligent assistant language skills). It is thus important to make existing grammars reusable and make the creation of new grammars simple.

[19] `http://cloud.grammaticalframework.org/`

7 Results

The main contribution of our work is an implementation of an open and extendable speech application architecture which supports controlled natural language based speech interfaces, as well as a set of grammars covering a diverse set of domains that can serve as building blocks and examples for future grammars. We have applied the developed stack and grammars to Estonian grammar-based speech recognition and believe that it enables engineers with limited knowledge of speech processing and grammar engineering to quickly build speech-enabled user interfaces.

Flexible solution for application builders The described architecture makes it easy to port existing speech recognition based UIs to new languages. For example, the Android app Speaktoit Assistant has been developed with only English speakers in mind. With no change to the original app, it can be made to accept another language. One only needs to implement a grammar that maps the commands in the new language to the format required by the application, and add a new acoustic model to the speech recognition server (if it does not already exist). Using the Android's intent mechanism, it is also possible to easily add a speech-based UI to an app that did not have it before.

Improved accuracy over statistical language models We compared the accuracy of the grammar-based recognizer with the accuracy of a wide coverage recognizer whose statistical language model is trained over mainly news texts and uses a lexicon of 200,000 non-compound wordforms. The error rate of using Google Maps with the *Direction* grammar was much lower than when using it with free-form input. The experiment was based on 100 audio recordings made by two speakers with a smartphone in a room environment. Each recording contained a second-long utterance of a Tallinn address in the form of *street-name house-number*. These addresses were selected randomly from a large set of Tallinn street addresses scraped from the web. The grammar-based recognizer transcribed 90% of the recordings correctly, while the free-form recognizer achieved only 60% correct results. It should be noted that the free-form recognizer used a general language model and a much larger vocabulary (which might give preference to words which are frequent in Estonian, but not necessarily in Estonian street addresses).

Scalability to a large number of terminals The *Direction* grammar also demonstrates that our speech recognition architecture can handle a large number of terminals without any slowdown in processing nor major negative effect on precision. The $\sim$6000 place names in the grammar cover most of the geographical locations in Estonia, i.e. such a grammar can be used as a component in a car navigation system developed for speakers of small languages.

Uptake of grammar-based solutions After six months on Google Play, *Arvutaja* has proved to be a relatively popular Android app in Estonia with an install base of $\sim$1300 users and $\sim$19,000 queries since the initial launch. This makes it also the most frequent client application of *Kõnele* whose install base is 3700.

8 Future work

There is a number of extensions to the described work which we want to explore in the future. It would be useful to combine controlled with free-form input, e.g. for utterances like "text Bob I'm running late", where only the first two words are controlled and mapped to a structured format, while the last part could be recognized using a statistical model and returned as an uninterpreted string.

In order to make grammar creation accessible to casual users, it must be made really simple. In some cases, it can also be completely automated, e.g. turning one's contacts list (which is usually available on the mobile device) into a grammar and making it available in speech applications via commands and queries like "call mom's work number", "driving directions to Mati's home".

Our current grammars are limited to being regular in expressivity because our speech recognition engine processes them with finite-state technology. We plan to investigate what are the most compelling reasons for using full GF expressivity in the context of speech recognition and intend to integrate GF better into our speech recognition engine. In order to obtain higher transcription precision with larger grammars we want to look into probabilistic GF grammars.

Providing a full set of "smart paradigms" for Estonian (see [8] for Finnish) would simplify the work of grammar developers, who then would not need to know the details of Estonian morphology. Also it would simplify the introduction of more variation into the grammars, e.g. 'viis jagatud kuus' and 'viis jagatud kuuega' are equally likely expressions of $5/6$, but the latter is currently not supported because it contains a non-nominative case ending.

Sometimes the application response is structurally more complex than a simple "setting of an alarm", e.g. an address query can result in a set of driving directions. In this case it can be useful to present the results in natural language and even in the form of speech. GF could again help to translate the application format into a format suitable to the speech synthesizer. The speech synthesizer input, in this case, is not necessarily an orthographic text but could contain the information that is available in speech, e.g. palatalization and vowel/consonant quantity degree in case of Estonian.

The recognition server query logs provide valuable data about the current usage of our Android apps. Analyzing *Arvutaja* queries for the most common causes of out-of-grammar failure would let us fine-tune the grammars.

Acknowledgments This research was supported by the Estonian Ministry of Education and Research target-financed research theme no. 0140007s12.

References

1. Tanel Alumäe. *Methods for Estonian large vocabulary speech recognition*. PhD thesis, Tallinn University of Technology, 2006.
2. Tanel Alumäe and Kaarel Kaljurand. Open and extendable speech recognition application architecture for mobile environments. In *The third International Workshop on Spoken Languages Technologies for Under-resourced Languages (SLTU'12)*, Cape Town, South Africa, 2012.

3. Krasimir Angelov, Björn Bringert, and Aarne Ranta. PGF: A Portable Run-time Format for Type-theoretical Grammars. *Journal of Logic, Language and Information*, 19(2):201–228, 2010. 10.1007/s10849-009-9112-y.
4. Björn Bringert. Speech Recognition Grammar Compilation in Grammatical Framework. In *SPEECHGRAM 2007: ACL Workshop on Grammar-Based Approaches to Spoken Language Processing, June 29, 2007, Prague*, 2007.
5. Harald Hammarström and Aarne Ranta. Cardinal Numerals Revisited in GF. In *Workshop on Numerals in the World's Languages, Dept. of Linguistics, Max Planck Institute for Evolutionary Anthropology, Leipzig*, 2004.
6. William Meisel. "Life on-the-Go": The role of speech technology in mobile applications. In Amy Neustein, editor, *Advances in Speech Recognition*, pages 3–18. Springer US, 2010.
7. Einar Meister and Jaak Vilo. Strengthening the Estonian Language Technology. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, 2008.
8. Aarne Ranta. How predictable is Finnish morphology? An experiment on lexicon construction. In J. Nivre and M. Dahllöf and B. Megyesi, editor, *Resourceful Language Technology: Festschrift in Honor of Anna Sågvall Hein*, pages 130–148. University of Uppsala, 2008.
9. Aarne Ranta. *Grammatical Framework: Programming with Multilingual Grammars*. CSLI Publications, Stanford, 2011. ISBN-10: 1-57586-626-9 (Paper), 1-57586-627-7 (Cloth).
10. Aarne Ranta, Krasimir Angelov, and Thomas Hallgren. Tools for multilingual grammar-based translation on the web. In *Proceedings of the ACL 2010 System Demonstrations*, pages 66–71, Uppsala, Sweden, July 2010. Association for Computational Linguistics.
11. Manny Rayner, Beth Ann Hockey, and Pierette Bouillon. *Putting Linguistics into Speech Recognition: The Regulus Grammar Compiler*. CSLI Publications, Stanford, 2006.
12. Rolf Schwitter, Anna Ljungberg, and David Hood. ECOLE — A Look-ahead Editor for a Controlled Language. In *Controlled Translation, Proceedings of EAMT-CLAW03, Joint Conference combining the 8th International Workshop of the European Association for Machine Translation and the 4th Controlled Language Application Workshop*, pages 141–150, Dublin City University, Ireland, May 15–17th 2003.
13. Nikos Tsourakis, Maria Georgescul, Pierrette Bouillon, and Manny Rayner. Building Mobile Spoken Dialogue Applications Using Regulus. In *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, 2008.