EPF Lausanne

Final Report for Optional Semester Project

Communication application for children with speech disabilities (for Android devices)

by Vidmantas Zemleris, M.Sc. in Computer Science

Supervision: Francisco Pinto

Fall 2011/2012

Introduction

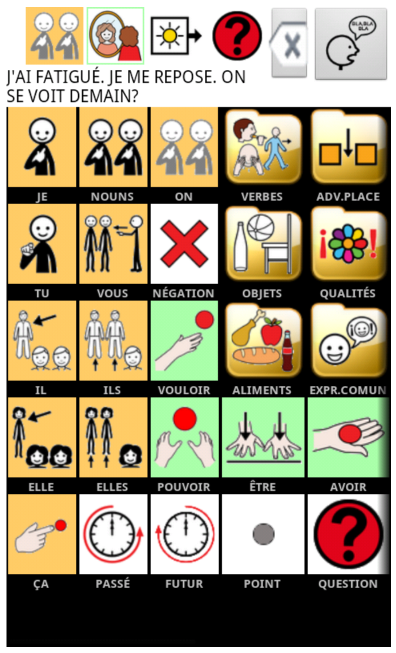
An Android application for children with speech disabilities has been created, that shall provide a generic and easy to learn communication method for anyone with speech disabilities. It forms sentences from a list of pictograms clicked/chosen. It is different from most of existing products by being free, by it's natural language processing features (inflection, tenses, using the setting user gender to alter first person sentences and more) and by usage of gesture search on mobile phones where keyboard is not reliable because of small screen.

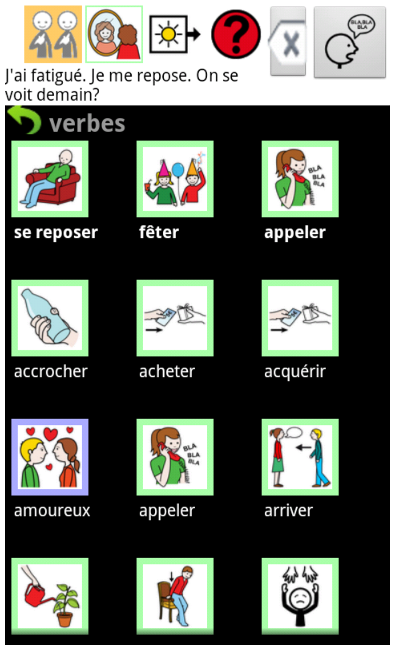
The application includes ~5000 icons obtained from arasaac.org, where ~900 are properly categorized, and others are accessible through search. It supports French language well, and there is an early prototype for English too.

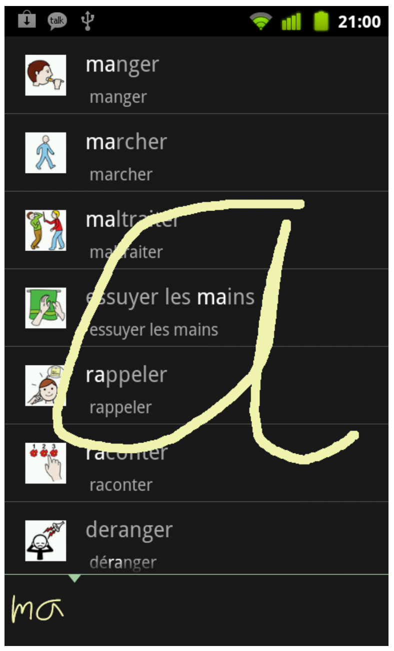
Other features include:

- recent phrases and pictogram history (each category lists most used ones on top)
- accessibility
  - for easier identification icons tagged with the 6 SPC colors (actions, names, descriptives etc)
  - option to display text in capital letters
- compatibility with both Tablets and mobile phones

Below are application screen-shots on mobile phone


J'AI FATIGUÉ. JE ME REPOSE. ON SE VOIT DEMAIN?


J'ai fatigué. Je me repose. On se voit demain?



home screen

category view (most used words on top in bold)

gesture search within category (by drawing letters)

Project Deliverables

As we were able to find a very comprehensive icon set, it has been decided to to process it (an unplanned task) so at the end of project we would have a fully usable application. Below is the list of project's work-products:

- Android Application for tablets and mobiles
  - available at: https://github.com/vidma/aac-speech-android/tree/master/source
- Items not planned initially
  - tools to process icon dataset
    - available at: https://github.com/vidma/aac-speech-android/tree/master/iconset-tools
  - icon set ready for usage (of limited quality, additional human processing needed)
    - including some icons manually assigned, modified or created: 17 category icons + 19 home-screen icons
    - available at: https://github.com/vidma/aac-speech-android/tree/master/content
  - developed a replacement for *Gesture search* that is not yet supported on Android 3.2 Tablets
  - rewriting of XML lexicon loading routines of *simpleNLG* library used in project
    - available at: https://github.com/vidma/aac-speech-android/blob/master/source/libs/simplenlg/lexicon/XMLLexiconFast.java

State of the art in AAC

A number of articles, books [Glennen, 1997][Beukelman, 2005] and various lecture notes in Alternative and Augmentative Communications (AAC further on) to have been reviewed get overview of the state-of-the-art of IT in AAC. Below we present a brief summary of most notable input methods and their applications in terms of performance or popularity.

Icon based – this is the most popular input method both for paper and electronic AAC tools *Pathfinder* – one of most popular devices; uses "semantic" combinations of buttons to represent words; it has been speculated that it provides one of the higher performances. However it's hard to learn [Emails with ARASAAC, 11], expensive (~7K to 15K $) and really heavy (2-3kg)

Existing *applications for Android* have no French support (Google/Android market) and/or are relatively expensive (200$ for iPhone app) and/or are limited in functionalities.

Keyboard based
Swype – user just swype's the finger through the keyboard on touch screen, and the application gets the best statistical guess of words corresponding to these phrases and also include error detection in the curves made. It has been claimed that experienced user could achieve 300 symbols per minute, that is a lot for mobile device. Indeed, from the authors experience, it works really great for English language, and fairly good for French.

Non-invasive sensor placed near the vocal chords captures the neurological signals that are sent to the muscles of vocal chords and these after processing are used to reconstruct the phonemes (the method is applicable only to ones who have these impulses).

Still the performance of this method shall be improved, but prototypes exist that provide very "naturally looking" dialogue with no noticeable external devices/interfaces needed [Audeo, 2011] for relatively low price[1]. This method being only a couple of years old, looks very promising and is clearly worth of further investigation.

Other more specific applications exist for people with motor disabilities who are unable to use other methods mentioned above include use of *proximity sensors*, *eye tracking*, *Brain-Computer-Interfaces, all having* relatively low throughput rate.

Therefore, it has been chosen to focus on icon-based (without *Pathfinder's* "semantic" combinations for now) method that is the most intuitive, and do not have any good replacements for Android.

[1] $2000 for sensor + research SDK

Application Architecture

Natural Language Generation from Pictorial Icons

We will take advantage of existing natural language generation systems (NLG) to generate grammatically correct language from as less of *pictographic* input as enough. That shall enable even children so young that have not yet developed their ability to write to produce generally sound sentences with help of this tool.

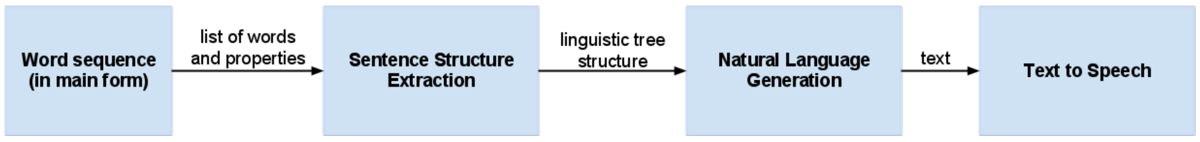

Word sequence (in main form)
list of words and properties
Sentence Structure Extraction
linguistic tree structure
Natural Language Generation
text
Text to Speech

Figure: pictogram to natural language processing flow

The NLG used *simpleNLG* will take care of complex French grammar including tenses, inflection, specific word-order etc . So what we need to do is to map the list of pictograms entered into a lexical structure (sort of tree of objects) to be fed to a NLG system (*simpleNLG*).

A bit simplified subset of French language may be considered as grammar close to CFG [Salkoff, 1980]. So, we could define a fairly simple CFG that is easy to parse and at the same time resembles the natural spoken French. We base this on [Salkoff, 1980]**,** simpleNLG documentation [simplenlg, 2011] and our notes on French grammar.

A CFG resembling the functionality implemented shall look like this (there are a couple of additional cases implemented):

```
NP = adj*  Nuon  adj*
PP = prep NP?
Subject = NP+
object = NP* |  adv* | adj* | PP
clause = subject     modal? verb   object*
sentence = (clause [.?!])+

definitions:
NP – noun phrase
PP - preposition phrase
adv – adverb
adj – adjective
prep – preposition
symbols based on regexp: * = 0..inf ;  + = 1..inf
```

Notice: for simplicity we do not support sub-clauses clauses yet. It could be implemented by matching words like *que, parce-que etc*, however most of phrases may be expressed as simple clauses anyway.

As we already have (simplified) part-of-speech tags associated with each pictogram, this is even easier task than parsing a Context-Free-Grammar and a simple linear “greedy” algorithm has been applied to generate this “object-tree” sentence structure.

Our (simplified) algorithm:

```
process pictograms in a loop:
       current = next not matched pictogram from left to right

        # match as many of NP in a row, but NP may contain only one noun + adjectives
        subject = first NP coordinate

       object =  match all subsequent NP coordinates;
                   if no specifier:                                         (e.g. je mange glace)
                       try to guess specifier (de, à) from the verb      (e.g. je mange de glace)

                 # match preposition phrase
                 if current=prep:
                      PP = Prep + match subsequent NP
                      clause.addComplement(PP)

                 if current=adverb:
                      clause.addComplement(all adverbs matched in a row)

                 # adjective attached to the clause, not to noun, e.g. tu es joli
                 if current=adj:
                      clause.setComplement(adjective)
       set verb
       if more than one verb:
               set first verb as modal  # e.g. je veux manger
       set tenses, negation and other features etc
```

To make the development easier, less error-prone, and to ease the change management, set of Unit tests () have been designed for our pictogram to natural language conversion code (nlg.*Pic2NLG* class).

Guessing prepositions (for NPs with no specifiers)

Some verbs usually go with one preposition more often than with others (e.g. *manger* goes with *de, while aider goes with à*) [About.com, 2011]. By guessing the most probable preposition (from the verb and what is following it) we eliminate extra input step.

Processing of ARASAAC icon set

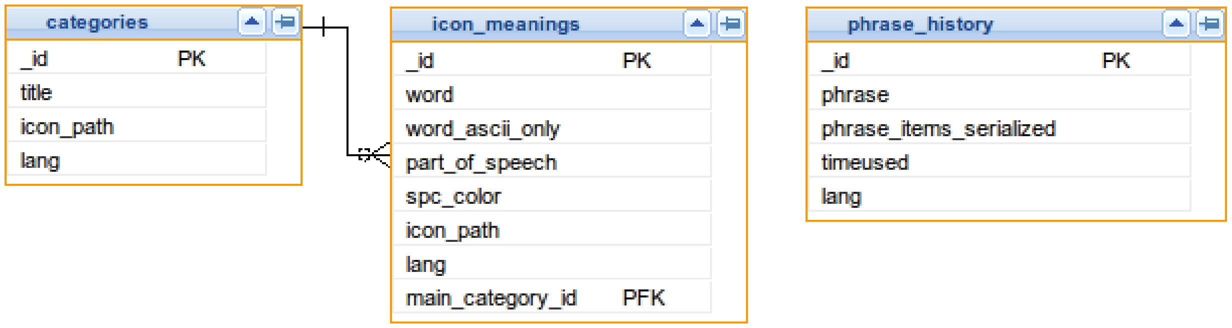
ARASAAC provided us with 5000+ pictogram images and a couple of CSV files that had to be pre-processed (resize images; remove the image/verb duplicates; transform the data model; add part-of-speech tags) before using them in the Android application.

A small python/django based application has been created to ease tasks of importing, exporting, management and search of icons. Later this could be easily extended to provide further functionality, for cleaning up the dataset, e.g. selecting proffered icons in case of collisions, reviewing the part-of-speech and category tagging etc.

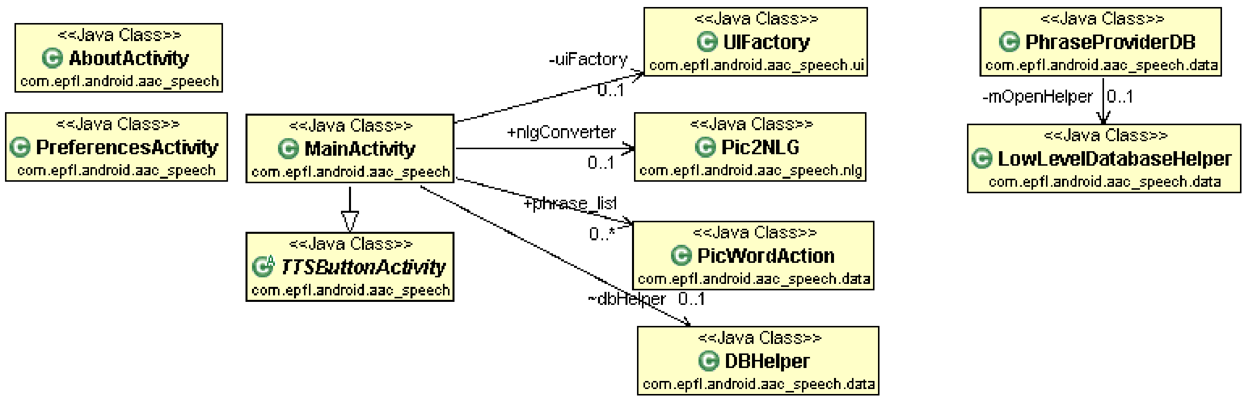
Technical implementation

Data model

App uses *SqLite* database. Data model is really simple for now. The current icon set did have only a couple of icons belonging to more than one category, so for now we stored it just as a foreign key. *TODO: improve categorizations and add a separate M-to-N relation table.*



Class diagram (most important classes only)



We have three Android Activities (i.e. screens/small-applications). The *MainActivity* uses a natural language generator (our *Pic2NLG* class), *UIFactory* to create UI elements including image buttons, and stores a list of currently selected pictograms in *phrase_list (each of type PicWordAction). DBHelper* provides database operations, like getting Cursor for items within category or saving history.

PhraseProviderDB is a content provider based on sqlite database that provides access (query, insert) to pictogram data through URIs (higher level of abstraction than just table names). These URIs are used in *managedQuery*() on *DBHelper*, and required by Google's Gesture Search library for accessing our pictograms dataset.

LowLevelDatabaseHelper takes care of database creation (creating/upgrading schema, parsing and importing data).

Some more interesting technical solutions used

Loading large number of Bitmaps; memory management

For category listings we use a *GridView* with a *aac_spech.ui.PictogramCursorAdapter* that loads the items on demand then scrolling. The images are loaded and resized from SDCARD on separate asynchronous thread[1] that provides huge increase in performance over loading images on UI thread (scrolling would be very very slow otherwise).

Usage of *bitmap.recyle()* and *WeakReference*[2] allows application not to crash even if it ran very low on memory resources (bitmap operations are expensive, while *simpleNLG* library takes lots of memory too).

Solving Issues with *simpleNLG* lib. performance and memory usage

Performance profiling with *DDMS tool* showed that the one of showiest methods during loading of simpleNLG are all related XML DOM parser.

Rewriting *simpleNLG's XMLLexicon* class to use *XmlPull*[3] instead of Node-based parser, we got loading time decreased by ~30%[4], and memory use have greatly decreased too, making scrolling of *GridView items with ImageView's* stable (scrolling was very laggy because of out of memory errors on HTC Desire mobile phone).

Compatibility with both Mobiles and Tablets

aac_spech.ui.HorizontalScrollView: allows updating and scrolling the current icons listing to end after it has changed. The contents of regular scrollview are not updated immediately, that's why we need additional *onLayout* listener.

aac_spech.ui.DynamicHorizontalScrollView: class provides a widget that allows to flip/swipe through screens (like on Android's launcher). By creating virtual screen space for smaller mobile screens, that allows to present the same content both to tablets and mobiles making it easier to support both.

View Switching with *ViewFlipper*

For performance reasons, instead of starting a new activity for each task (it's expensive: new process + inflating widgets + more complex inter-activity communication through intents only), we use V*iewflipper* to switch between home-screen, categories and search views.

Downloading icons as zip file and "pipelined" unzipping

As we have ~5000 icons making 20+MB, downloading each individually would be disaster. The icons are downloaded to the storage location on the device (sdcard for mobiles; internal sdcard for tablets).

MergeCursor is used to easily display[5] the most used items in the same GridView as the category items are: we join two Cursors, one for most used items within that category and other for all items in the category.

[1] though setting of the image resource has still be done on UI thread, as only UI thread is allowed to access

[2] WeakReferences allows the garbage collector to dealocate referenced objects (the bitmaps scheduled to be assigned to imageviews, where some of them may be not relevent anymore because of scrolling) if application went really short on memory.

[3] http://www.xmlpull.org/

[4] from 24s to 17s on HTC Desire with 1Ghz processor

[5] Note that it's not possible to put two separate GridViews inside a Scrollview, as scrollview would capture the UI events and it would require to manually layout the gridviews (measure they height).

Distribution

The application will be submitted to Android Market as beta app. ARASAAC would also advertise our application. Application's source code has been made freely available at *GitHub* at [aac-speech-android, 2012] and application shall continue to evolve through open-source.

To ease testing and error reporting, a unhandled errors reporting (*StacktraceSubmitExceptionHandler*) could be used (but is not yet activated – need to set up a server etc) that would send a notification to our server upon an unexpected exception (e.g. lack of memory issues[1]).

Also, in case of such unexpected errors the application would restart instead of just crashing in nasty-way.

Future work

- clean up the icon-set further
    - see if better categorizations are possible
    - provide better part-of-speech tags
    - remove duplicate icons so that only best icons are selected for each word (currently any icon is selected)
- extend/improve natural language processing
    - automatic guessing, compound words: avoir besoin etc, personal pronouns (me, te, etc)[2], add other question and negation types (ne..que, ne..guerre etc, in addition to *ne .. pas*)
- extend user interface
    - more options for word icons (e.g. overriding number/gender for each icon)
    - better UI for tablets: have filtering options on the side of category view
- could we further speed up the loading time?
- *Gesture search do not handle French accents yet (bug in Google's app)*
    - currently we use the ascii version of French word for search
- Implement English

Conclusions

Because of it's natural language processing features, after finishing the work left it's going to become a fully featured one of the most advanced applications on the market, and it's free.

Further interesting topic worth to research is on the non-invasive sensors next to vocal chords (targeted at a subset of ones who may use the application developed now) that would provide almost seamless assistance, and is closest to the natural speech.

[1] Android applications have limit on heap size of 16/24 MB (depending on version). In addition to what we already do (see section on 'memory management & pictograms') there are a couple ways around that, but they're not so easily applicable.

[2] At the moment these work properly only if selected after the verb

References

[aac-speech-android, 2012] https://github.com/vidma/aac-speech-android/

[About.com, 2011] http://french.about.com/library/prepositions/bl_prep_a_verb.htm

[Audeo, 2011] http://www.theaudeo.com/?action=technology

[Beukelman, 2005] David R. Beukelman, Pat Mirenda, Augmentative & alternative communication: supporting children & adults with complex communication needs, Paul H. Brookes Pub. Co., 2005

[Emails with ARASAAC, 11] Email discussions with Administrator David Romero and AAC specialist José Manuel Marcos of "Aragonese Portal of Augmentative and Alternative Communication" http://arasaac.org/

[Glennen, 1997] Sharon Glennen, Denise C. DeCoste, The handbook of augmentative and alternative communication, Cengage Learning, 1997

[Salkoff, 1980] http://aclweb.org/anthology/C/C80/C80-1007.pdf

[simplenlg, 2011] http://code.google.com/p/simplenlg/wiki/Section3

[simplenlg-en-fr, 2011] http://www-etud.iro.umontreal.ca/~vaudrypl/snlgbil/SimpleNLG-EnFr_doc_francais.pdf