UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

Customizable vocal personal assistant

For Windows10

Dissertation presented by
Tanguy DE BELS

for obtaining the Master's degree in
Computer Science

Supervisor(s)
Charles PECHEUR

Reader(s)
Kim MENS, Gorby KABASELE NDONDA

Academic year 2016-2017

Acknowledgments

I would like to thank my supervisor Charles Pecheur who accepted the project I submitted to him.

I would also like to thank my beta-testers crew which I tortured with every new version of the program even though they avenged themselves by gladly pointing out what did not work. Of course the same applies to my proofreaders.

And like for every project, big thanks to OpenClassrooms which supported me during my whole studies.

Abstract

Since 2007 and the introduction of SIRI, there are more and more intelligent personal assistants appearing. The enthusiasm comes from their ability to manage several aspects of our life. Some of them such as Siri or Cortana are developed by big companies but there are also others developed as open-source or individual projects like Jarvis or Lucida.

Even though they gain more and more capabilities with the years and upgrades, they are still eventually hitting the limits of the software. We are limited by which functionalities they allow us to run and the programmed capabilities. The same goes for the social capabilities that are limited to encoded answers that we can not expand. The focus of this work is to go beyond these limits and allow the user to customise his own assistant with functions responding to his needs.

We will first proceed to the creation of a new intelligent personal assistant with basic functions. We will then add a macro recorder to enable an user to add functionalities to the programmed ones. This whole recording process should be feasible by no-programmer users.

Contents

List of abbreviations and definitions

IPA: An *Intelligent Personal Assistant* is a software agent that can perform tasks based on user input, location awareness, and the ability to perform various queries on large scaled databases.

Macro: Generic term designing a mean to memorize a sequence of tasks inside a software with the goal to use it to repeat the sequence of tasks.

Phones: In phonetics and linguistics, a phone is any distinct speech sound or gesture, regardless of whether the exact sound is critical to the meanings of words.

GMM: A *Gaussian Mixture Model* corresponds to the mixture distribution that represents the probability distribution of observations in the overall population. The Gaussian one are among the most statistically mature methods for clustering (though they are also used intensively for density estimation).

ANN: An *Artificial Neural Network* learns (progressively improve performance) to do tasks by considering examples, generally without task-specific programming. Neurons are organized in layers. Different layers may perform different kinds of transformations on their inputs. Signals travel from the first (input), to the last (output) layer, possibly after traversing the layers multiple times.

DNN: A *Deep Neural Network* is an ANN with multiple hidden layers between the input and output layers. DNNs can model complex non-linear relationships. DNN architectures generate compositional models where the object is expressed as a layered composition of primitives. The extra layers enable composition of features from lower layers, potentially modeling complex data with fewer units than a similarly performing shallow network.

HMM: A *Hidden Markov Model* can be considered a generalization of a mixture model where the hidden variables, which control the mixture component to be selected for each observation, are related through a Markov process rather than independent of each other.

Lemmatisation: In linguistics is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form.

Stemming: Is the process of reducing derived words to their word stem, base or root form, generally a written word form. The difference with lemmatisation can be illustrated by the word "better" which has "good" as its lemma. This link is missed by stemming, as it requires a dictionary look-up.

List of Tables

List of Figures

Chapter 1

Problem

1.1 Origin of the project

Upon the release of Microsoft IPA, Cortana, I decided to migrate to Windows once again because I was attracted to the concept and wanted to experiment it myself.

As soon as I had installed Windows 10, I was asking to Cortana to play music or tell me jokes. A great tool indeed but I quickly found the limits of its possibilities. It may sound silly but when you are lying on the couch and would like to decrease the sound of your computer. An IPA which have the correct function to answer your order may come in handy.

It was the offspring of the idea. As a programmer it was not be so hard to design functions responding to my needs. All I needed was to do the mapping with a few command-sentences. Then when I added speech recognition and speech synthesis I had my own personal assistant!

After showing off a bit, friends started to ask to have their own tailored. In other words: more work for me. Would not it be amazing if they could customize it by themselves? This question was the start of this master thesis.

1.2 Objectives

- *Develop a vocal IPA.* It should be able to perform the basic functions of this kind of software such as internet search, launch music, read mails, ..
- *Integrate a macro recorder.* Design an user-friendly way to run this macro recorder to add functions to the predefined ones.
- *Make the macros parametrizable.* The text part of the recorded behaviours should be altered based on users wishes to customize functions.

1.3 Approach

Objectives	Approach
Develop a vocal IPA	• Choose the most suited speech recognition software/library. • Choose the most suited text-to speech software/library. • Design and implement functions usable by the IPA. • Design and implement a mapper to recognize which requests are asked by the user with his orders.
Integrate a macro recorder	• Choose the most suited macro recorder software/library. • Implement an user-friendly way to register macros and add it as functionalities callable by the IPA.
Make the macros parametrizable	• Design an automatic modifier of the macro files. • Incorporate the choice of parameters to the execution of the macros.

Table 1.1: Objectives steps approach

The table above describes all the achievements that had to to be reached for the completion of each objective. It is important that even though the objectives needed to be completed in the given order, each of the parts constituting it were doable in any order. This is due to the modularity of the software developed.

1.4 Roadmap

This work will be articulated around the conception of the IPA. We will first start with a quick summary of the material used for the conception and the related works. This document will then discuss the design choices made to make this software innovative among other existing IPA.

These points will be illustrated on the next section with running examples. The validation part of the software will then be discussed. This part will lead to the conclusion and the possible improvements.

Remark: It is important to notice that some examples will be illustrated in French instead of English because of a design choice explained in the subsection 3.1.2.

Chapter 2

Background information

2.1 State of the art

2.1.1 IPA Concept

A virtual assistant is a software agent that can perform tasks or services for a person. As an agent it uses the input, received from its sensors, as data to process before taking actions with its actuators. To illustrate the general functioning of an IPA we picked the Sirius model (the Linux clone of Siri) discussed in [1] to demonstrate the working of such an agent:

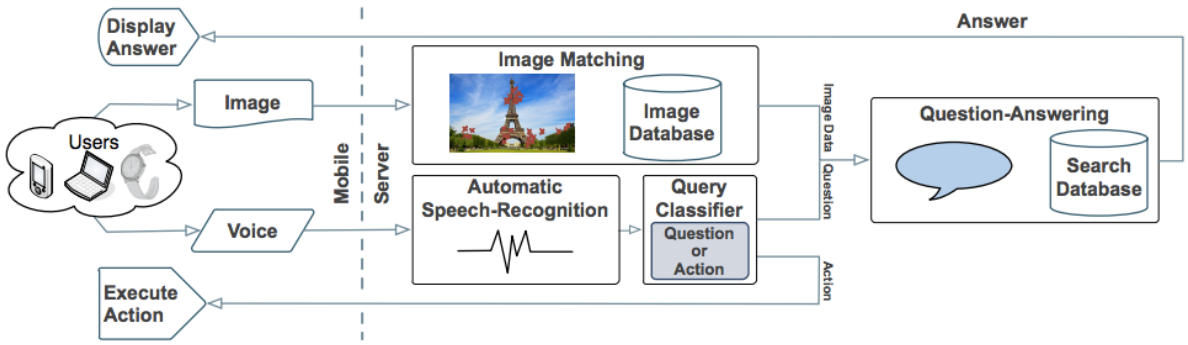


Figure 2.1: Sirius end-to-end pipeline

We can easily picture an agent in the schema above. The sensors are picking informations through the voice of the user and images. Actuators allow to display an answer or to take an action. The processing of the input and the answering are part of the analysis done by the agent before answering and taking actions.

This structure is the model on which the prototype is based, though our prototype will have to work with a far less populated database and less queries due to a lack of funds. This last point should be compensated by the capability to create our own dialogue answers and actions. More informations regarding agents can be found in [2].

2.1.2 Speech synthesis

Speech synthesis is the artificial production of human speech. Synthesized speech can be achieved by concatenating pieces of recorded speech previously stored in a database. Of course this would not provide an appreciable result without enough variety. The more phones, any distinct speech sound or gesture, the system is storing, the better the speech would be. The most adapted answer can be found more effectively with a larger set of phones. It is due to the fact that to evaluate the quality of a TTS system is judged by its similarity to the human voice.

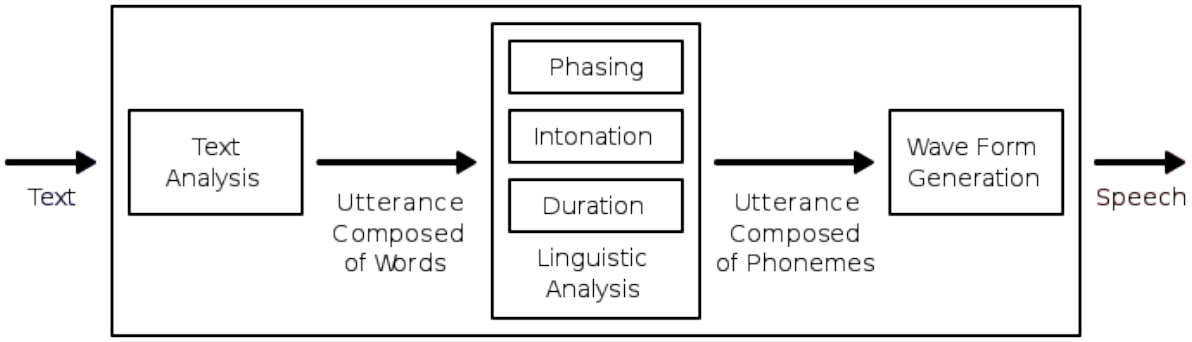


Figure 2.2: Typical TTS system

As we can see the TTS is divided in two parts: the first one converts raw text into the equivalent of written-out words (tokenization). The second one then assigns phonetic transcriptions to each word, and divides and marks the text into units, like sentences.

The treatment of the language used in this system was an inspiration for the work. The way the words are treated will be reversed and used for more generalization in the recognition of commands. But this point will discussed further in subsection 3.1.3 of this document.

2.1.3 Stemming

As seen in course [3], the manipulation of words enables more flexibility while interpreting a sentence. Stemming consists in the process of reducing inflected (or sometimes derived) words to their word stem, base or root form. Usually related words map to the same stem.

This last point is the interesting part of this technique regarding this prototype, stemming will allow it to search through a less populated data structure and then gain time. For example any verb would be reduce to its present form and command words would match with any time form used in a sentence.

A more complex and possible approach was lemmatisation. This process involves first determining the part of speech of a word, and apply different normalization rules for each part of speech. The part of speech is first detected prior to attempting to find the root since for some languages, the stemming rules change depending on the part of speech of a word. Since this approach is highly conditional upon obtaining the correct lexical category and was more time costly, it was not chosen.

2.1.4 Speech recognition

There is a field of computational linguistics that develops methodologies and technologies that enable recognition and translation of spoken language into text by computers. As explain in [4] this is computed by extracting the main features of the voice to retain vectors. They will then be classified according to the maximum likelihood criteria to give the assumed words corresponding to the speech, as shown in the figure below.

In speech recognition, the hidden Markov model will output a sequence of n-dimensional real-valued vectors (consisting of coefficients obtained by a Fourier transform of spectrum of the speech), outputting one of these every 10 milliseconds (to enable to approximate speech as a stationary process). The HMM will tend to have in each state a statistical distribution which will give a likelihood for each observed vector. Each phoneme/word will have a different output distribution. A HMM for a sequence of words or phonemes is made by concatenating the individual trained hidden Markov models for the separate words and phonemes.

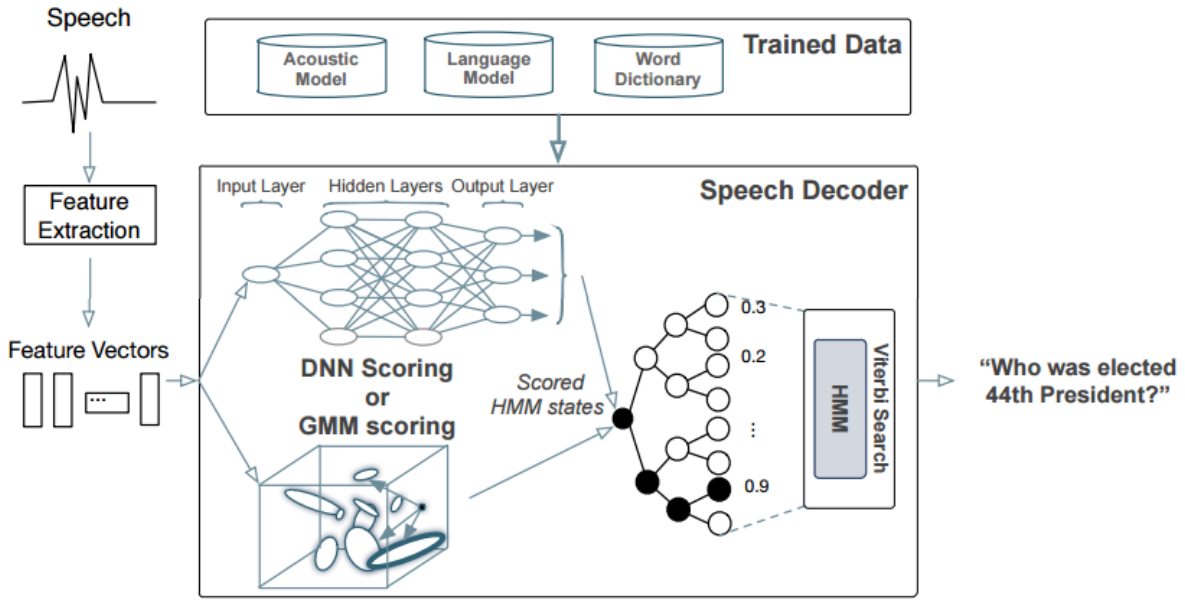


Figure 2.3: Typical Speech Recognition pipeline

According to [5], the use of a classifier based on the main features of the voice is particularly important for people with an accent or not talking in their native tongue language. We will discuss later in the subsection 3.1.2 this point because it had a huge impact on the project.

2.2 Related work

In the following table we can see a comparison of the capabilities of three chosen IPA. It is important to notice that none has the ability to add a customized function to its set of available ones, which is the innovation and the focus of this work. Nonetheless they have some great abilities, notably linked to the location which our prototype has not (yet) and could hardly be added by a non-programmer even with the macro recording system.

It is also important to notice that, according to [6] and practical use, only Google Now is always listening, while most of the IPA need an action to trigger the agent. This point will be further discussed in the implementation choices of this work.

	Cortana	Google Now	Siri	Our prototype
Possible to summon with hardware button	Yes	No - always listening of 'OK Google'	Yes	No - always listening once started
Web search	Yes	Yes	Yes	Yes
Geofencing (e.g. reminding you to purchase when you're near a business)	Yes	Yes	Limited	No
Predictive notifications (e.g. traffic on your commute is bad)	Yes	Yes	No	Yes - with customized function
Event or contact based notification (when you sister calls, tell her happy birthday)	Yes	Yes	Yes	Yes - with customized function
Answers sassy questions like "Are you sexy?"	Yes	No	Yes	Yes - with customized function
Add customized functions	No	No	No	Yes

Table 2.1: IPA capabilities comparison

2.2.1 Big data analysis

We figured it would also be important to point out a big difference between the IPAs from this work which is the use of big data by Cortana. Because of the large resources of Microsoft and their wide panel of applications and devices, they were able to create a more context sensitive software.

Cortana Analytics takes advantage of machine learning, unlimited data storage, and perceptual intelligence to transform data into intelligent actions. Even though it is mostly used by companies, it is highly relevant to comment that the use of big data allows to further improve every decision that impacts all aspects of our life and even without any customized functions, it is an IPA which will probably greatly evolve in the years coming.

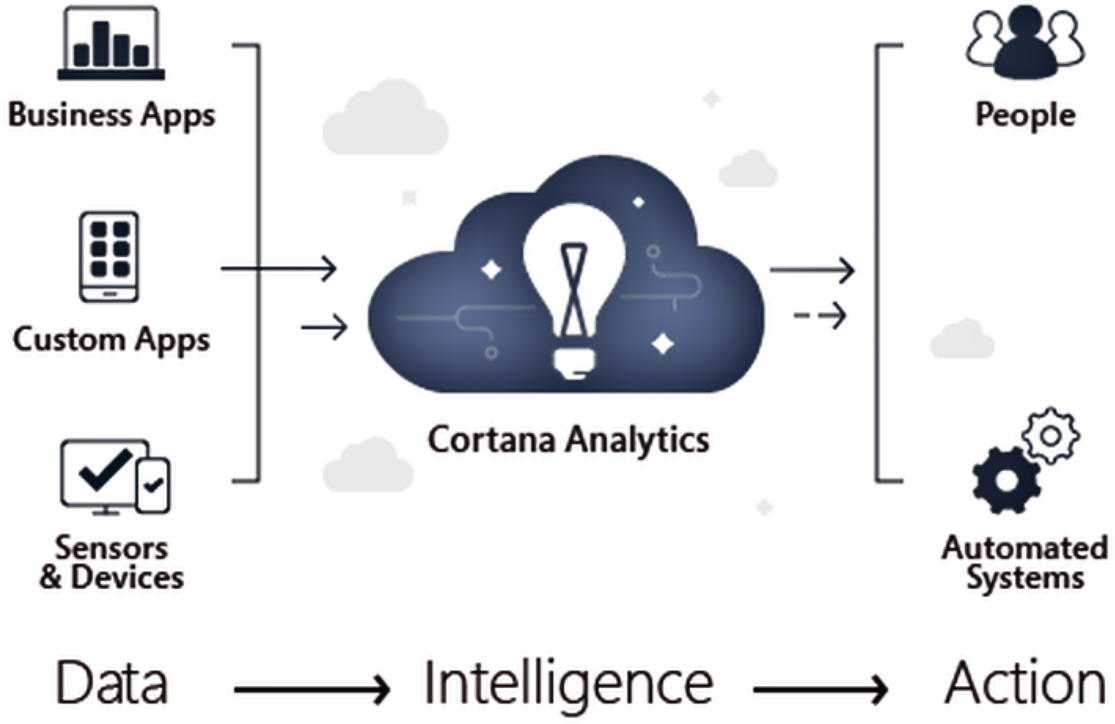


Figure 2.4: Cortana analytics

As it is shown in the figure above we can see that the input data is coming from the multiple applications and devices using Cortana. According to the data of [7] it represents for Microsoft a huge part of the OS market since it is still the most used OS on computers but of course the contribution from the smartphones is minimal because Microsoft is still not as implanted as other companies on this market.

Chapter 3

Conception

3.1 Implementation choices

After the analysis of the technologies and the same type of software, some choices were made to design the prototype of this work. The main choices will be briefly explained in the following section.

3.1.1 Speech recognition

Several approaches were tried for this particular point of the work:

- Recognition using a comparator between recorded command words / sentences (for example the user says once "hello" and his voice is recorded and stored locally) and recently produced one (the next time a user speaks his speech will be compared with the previous records after suppression of the noise).
- Google Speech Recognition API enables developers to convert audio to text by applying powerful neural network models in an API.

Google was finally chosen against the other solution although the other one had the benefit of being locally done since the records would be related to the user and stored on his computer. This was chosen since it allowed more flexibility and a sounder solution because it would recognize any text and not just test the likelihood with an already recorded sound.

This choice also means that anything said to the IPA is sent and recorded for the sake of improvement. This point in particular is annoying regarding the privacy but no way to solve it in a satisfying way was found.

3.1.2 A French IPA

As announced earlier, due to the use of a classifier to recognize the most likely pronounced words, we are extracting the main features of the voice. Although these features and the performances of the classifier can be altered by the fact that an user could list, sniffle or even just have an accent. This is due to the fact that the phone of a word can be really close to another one and the transcription is based on words and not sentences which would need more process and computation time.

This last point in particular led this work to a french orientation in its words recognition (to remove the accent issue) but also in its answers and in the way the text would be analysed and treated syntactically speaking (since the syntactic rules are different from a language to another). This last point in particular will be further explored in the next subsection.

3.1.3 Command recognition

A summary of the algorithm working on the recognition is needed to understand the concept used in this section: Command words or phrases are used in a hashmap as keys for the functions.

When a new sentence is heard, each word will be tokenized and then stemmed (process reducing the word to its root) to avoid processing the same root more than once.

To be able to stem in French, this project uses the Snowball French stemmer from the NLTK library. Snowball is a small string processing language designed for creating stemming algorithms for use in information retrieval. More precisely, the french stemmer removes upper-case, accented forms, suffixes, plural and other unnecessary informations and apply some replacement to obtain a more generic form such as "ç" with "c" or "y" with "i".

Here is an example of stemming from an IT forum [8] to understand how it works:

```
>>> from nltk.stem.snowball import FrenchStemmer
>>> stemmer = FrenchStemmer()
>>> stemmer.stem('voudrais')
u'voudr'
>>> stemmer.stem('animaux')
u'animal'
```

Figure 3.1: Stemming example

This allows more flexibility in the understood sentences. The keys in the hashtable are stemmed too so we do not need to pay attention to the forms for verbs, plural words and even some abbreviations.

There is another major improvement regarding the command recognition compared to other IPA. Indeed, it is possible to give several orders in one sentence instead of one order by sentence. Even better: these orders will be executed in the order they appear in the sentence. The algorithm leading to this will be explained in the section 3.3.

3.1.4 Always listening

In to make a more a more useful IPA it was decided to make it always listen by default. People tend to forget that they have the option to use their IPA and do not do it. Because it does not make it spontaneous to have to push a button or remember a code phrase to activate their IPA.

Regarding this point another approach was adopted. The IPA will always listen and try to recognize command words/sentences and will take actions if it is the case. The wish was that it would more interactive for the user.However this could bother people, especially regarding the privacy issue that was already exposed earlier. There is of course a function permitting to stop the IPA by voice.

Another option could be to make it awake for a specific command (like "Hey Cortana" for the IPA of Microsoft) but it would not change the fact that the voice would be recorded by the SR software. The only utility of this specific function is when the user does not want to hear the IPA when he is in the middle of something (talking to himself for example).

3.1.5 Speech synthesis

For this prototype the pyttsx library is used for speech synthesis. The output will not be as good as with a purchased TTS such as Ivona [9] but it will perform adequately enough.

3.1.6 Data storage

Hashtables are stored in text files using the pickle module. These tables also allows to easily update keys and associated functions. This is locally stored because even if it was hosted online the customized functions and the data related to it have no interest in being hosted elsewhere than on the user machine. But this point will be discussed further in the improvements section 6.2.

Pickle over json

- Pickle is a binary serialization format (instead of text one of json). Which means less storage used.
- JSON, by default, can only represent a subset of the Python built-in types, and no custom classes; pickle can represent an extremely large number of Python types.
- JSON is human-readable, while pickle is not; JSON is also interoperable and widely used outside of the Python ecosystem, while pickle is Python-specific.

Pickle over marshal

- The marshal serialization format is not guaranteed to be portable across Python versions. Because its primary job in life is to support .pyc files, the Python implementers reserve the right to change the serialization format in non-backwards compatible ways should the need arise. The pickle serialization format is guaranteed to be backwards compatible across Python releases. It is an important point regarding the sustainability of the software.
- The pickle module keeps track of objects it has already serialized, so that later references to the same object will not be serialized again.
- Marshal cannot be used to serialize user-defined classes and their instances. pickle can save and restore class instances transparently.

To store these tables, it was decided to use pickle over the two other solutions for the previous reasons. But it could change if the application grows further and more interoperability is needed, then JSON would be adopted.

3.1.7 Macro recording

Macro recorder

The choice was made to use a third-party macro recorder (so miscalled "Mouse Recorder") over an event-programming language because of its user friendliness for casual computer users. Even more this software allows to register macro of anything that an user is able to do with mouse and keyboard. It leads to more capabilities than what is available with the earlier mentioned languages (especially for non programmers).

Parameters

During the macro recording process the user will be able to record values that could change between his uses of the macro. The user will be asked to encode a default value given by the IPA. This value (easily recognizable) will be changed in the macro file before execution.

For the execution there was another question regarding the way the parameters would be asked to the user. The first attempt was to let the user create a command sentence with markers for each parameters inside. This allowed pattern-matching for the queries with the values of the parameters easy to figure out of the sentence. Here is an example:

Send email to param1 with param2 as object...

⇑

Send email to Friends with Got top grades for my TFE as object...

Figure 3.2: Use of a macro - First idea

Even though this method was elegant it had several drawbacks: first it was going against the will of making it as flexible as possible because the user would have to almost remember the exact structure of the recorded command sentence. Secondary it was less easy to tutor the user during the process and it would be less user friendly.

The final choice was to ask the user to give a name to each parameter during the elaboration process of the macro and then while he uses the macro to ask the value wanted for each parameter. This allows more freedom in the command word/sentence recognition. This enable the same flexibility than with the basic functions commands. The trigger can only be based on one word and upon the execution of the associated function, the parameters will be asked.

The final result would be looking like in the figure below with the same desired function:

"Send email please"
To whom?
"Friends"
What is the object?
"Got top grades for my TFE"

Figure 3.3: Use of a macro - Final idea

It is important to notice that the degree of human likelihood for the parameters questions is depending on the user. The first choice to make the IPA ask for "What value do you wish to give for the param parameter" but it sounded too mechanic and it was decided to let the sentence up to the user. That means the user either hear a nicely asked question or just a word.

3.2 Architecture

3.2.1 Agent structure

In the following figure you can see the structure of the prototype. As it was mentioned earlier it is based on the Sirius implementation.

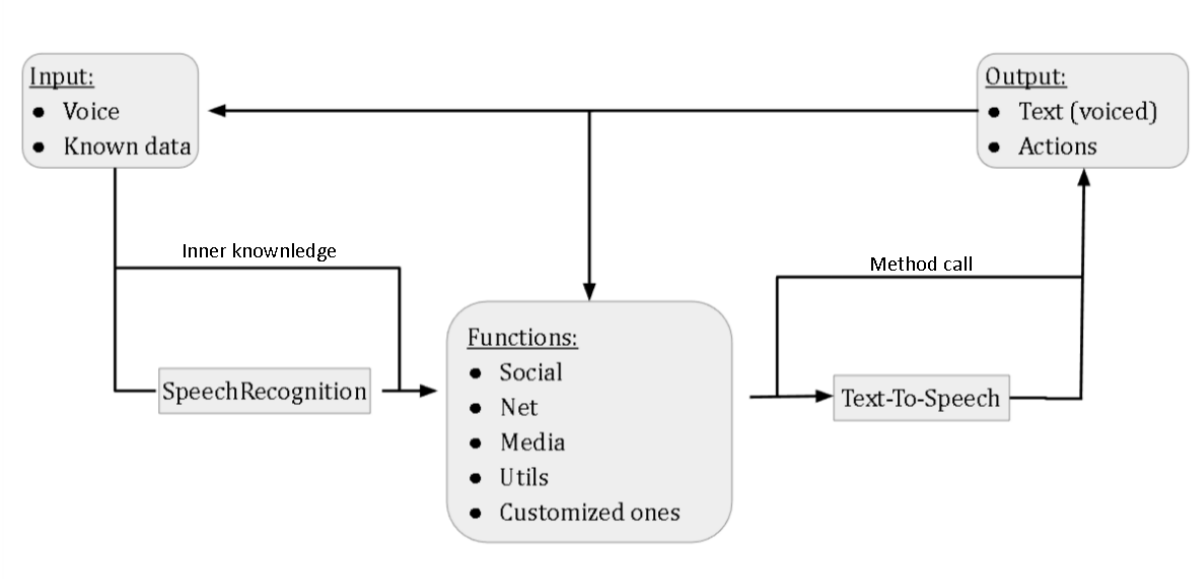


Figure 3.4: IPA agent structure

It is important to notice that the actions can affect both the future input and the functions. By modifying data it is possible to alter the known data which cant trigger another action later (the easiest example would be an appointment in the calendar). But the core of this program is to add functions and in that context actions allow to produce new functions and to add and store them for later use.

For the input event, even if the prototype does not deal with images as Sirius. It still uses speech recognition to match words to the voice and can be based on previous knowledge to trigger actions (including choices made in the macro definition).

This figure also shows that not all functions are using the TTS software. Some just trigger actions without any text being said (For example: launch an internet search).

The available functions are separated into files for the sake of modularity in the case of interoperability but this point will be discussed later in the improvements section.

- **Social:** Contains the social answer functions of the IPA such as saying "hello" back.
- **Net:** Contains all the functions using the Internet, from a basic search to displaying a specific website.
- **Media:** Contains the functions used to play multimedia contents.
- **Utils:** Contains the functions not fitting in any other module.
- **Customized ones:** Contains the functions needed to create, store and run the customized functions.

3.2.2 Data structures

In the following figure you can see the data structure of this prototype, with two main structures out of the core program:

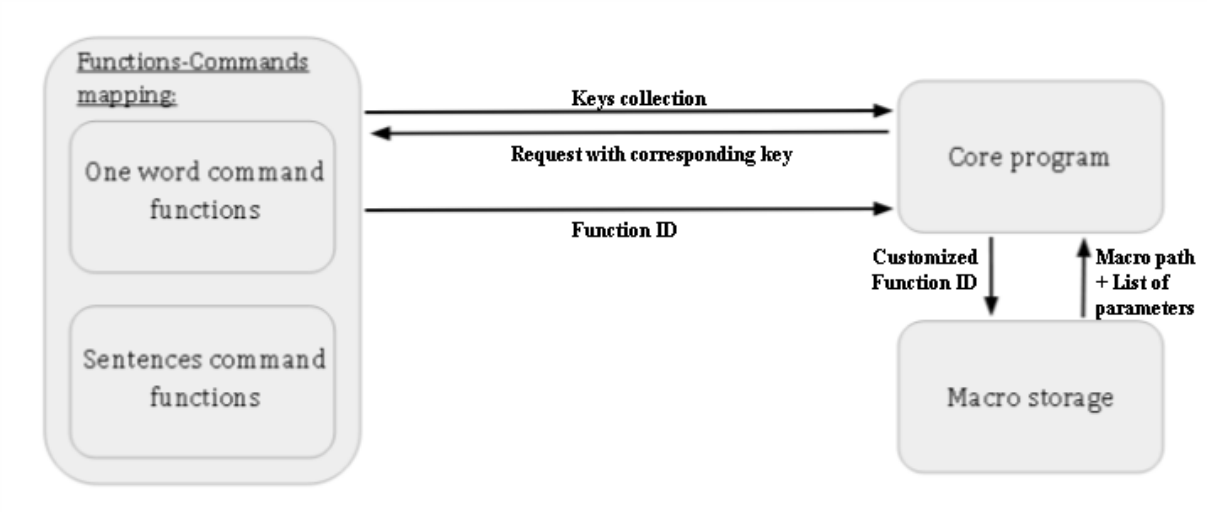


Figure 3.5: IPA data structure

The keys of the hashmaps mapping command words/sentences and the function are actually the command words/sentences. The distinction between one-word and sentence command is really important since it allows to more efficiently know if a sentence contains or not command needing to trigger a function. This point will be further explained in the algorithms section 3.3.

The choice for hashmaps is obvious because it allows access with a complexity of O(1).

The macro storage does not contain only a path to the macro to execute but also the list of parameters that were attributed to this macro when the user designed it. With this we can ask directly the wanted values before running the macro. The structure is of a hashmap with a value containing another one (in light blue), containing the path and a list of the parameters (in dark blue):

{(command word/sentence : {'path': path, 'params': [...]}) , ...}

Figure 3.6: Macro information - Data structure

3.3 Algorithms

3.3.1 Speech treatment

One of the main focus of this work, before even making any customization possible, was to make a decent IPA. The meaning behind these words does not lay in the number of actions it could achieve. The goal was to make it user-friendly and closer to an human in terms of social interaction.

And that is what is lacking for most of the IPA. While they treat the input, they are only looking for one action to execute out of the input. But this leads to a mechanization of the interaction. The following algorithm schemas show these issues are dealt with in this work.

One of the main issues was the impossibility to determine which commands were contained in the input sentence by doing only one run on the words given. It was due to the fact that commands could be more than one word.

For group of words commands, we would need to run and test through the given sentence with a size of one word then two words and continue to increment until we reached the current maximum number of words for a command. It would mean a big loss on computational power and time with big sentences which would not necessarily use this command. For example if "comment ça va" was a key, it would need three runs on the complete sentence to be able to match the key. Even worse it would compares the composed keys with several runs even when no composed key was used.

The alternative to keep a decent complexity was to test if keys were contained in the sentence instead of testing if a word or group of words was a key. The drawback of this solution was losing the order in which the commands were given.

The final decision was to make a trade-off between respecting the order and efficiency. The approach is clearly treating two cases differently: the one-word recognition from one part and the group of words recognition.

- **One-word recognition:** is done on each word of the sentence to know if it is a key. This allows only one run and the complexity is not too big because of the use of hashmaps.
- **Group-of-words recognition:** for this one we check if this particular set of keys have a key contained in a group of words. Even though the process could cost process time for no reason it is limited to one run only and a specific set of keys.

Although it would be a shame if all the group-of-words commands would be executed at the start or the end of the executed functions. There would not be much interest to only keep the order for the one-word commands. For example "Bonjour, commment ça va" could answer "Bien et vous? Bonjour." if the second treatment is executed first (the inverse is also true) which would be an issue regarding the order. This work is using the following solution:

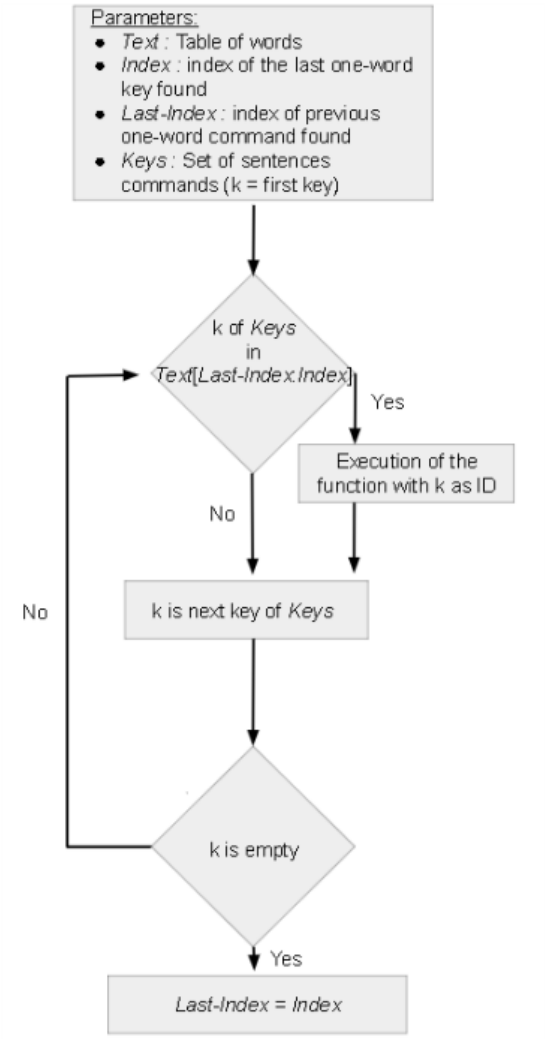


Figure 3.7: Group-of-words recognition algorithm

The group-of-words commands are only tested between index. These index are the last one-word command found in the sentence and the one before that. With this method we keep the order in the execution of the functions.

It means that for "Comment ça va, merci" even if the word "merci" will be recognized first, we will take its index and look if there is no group-of-words key before it in the sentence. "ça va" will be found and treated before the one at the index which ensure that we keep the order.

Remark:
The code of the algorithm is a part of the one available in appendix A.1

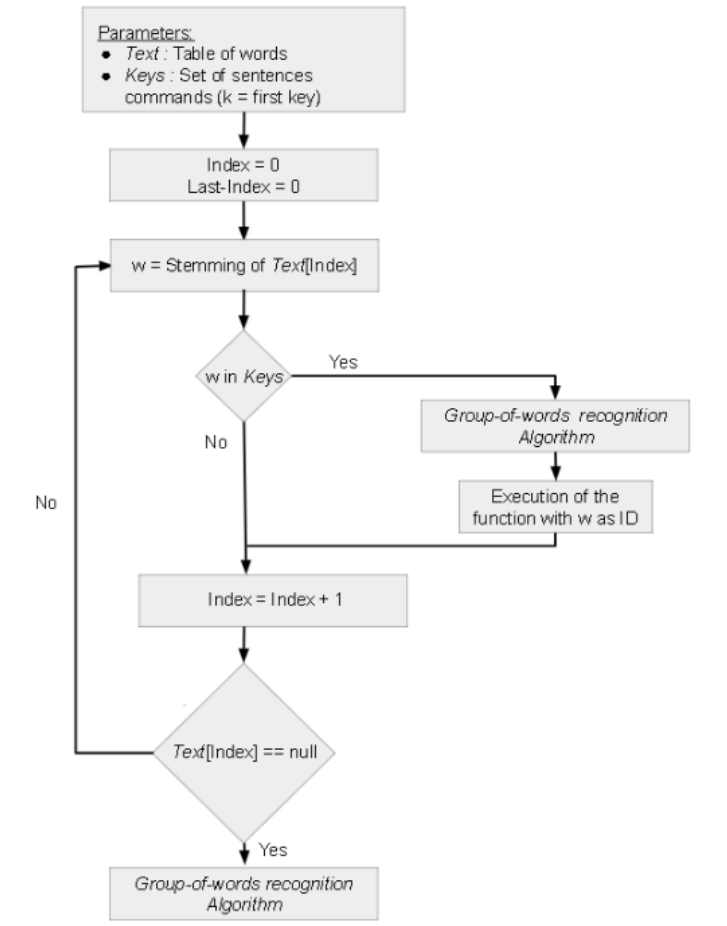


Figure 3.8: Commands recognition algorithm

The function of the last one-word command found acting as an index for the *Group-of-words recognition algorithm* is executed after the functions triggered by the other algorithm to keep the order as much as possible.

If no one-word command is found we still run the *Group-of-words recognition algorithm* since the absence of one type of command does not imply the absence of the other.

Remark:
The code is available in appendix A.1

3.3.2 Macro treatment

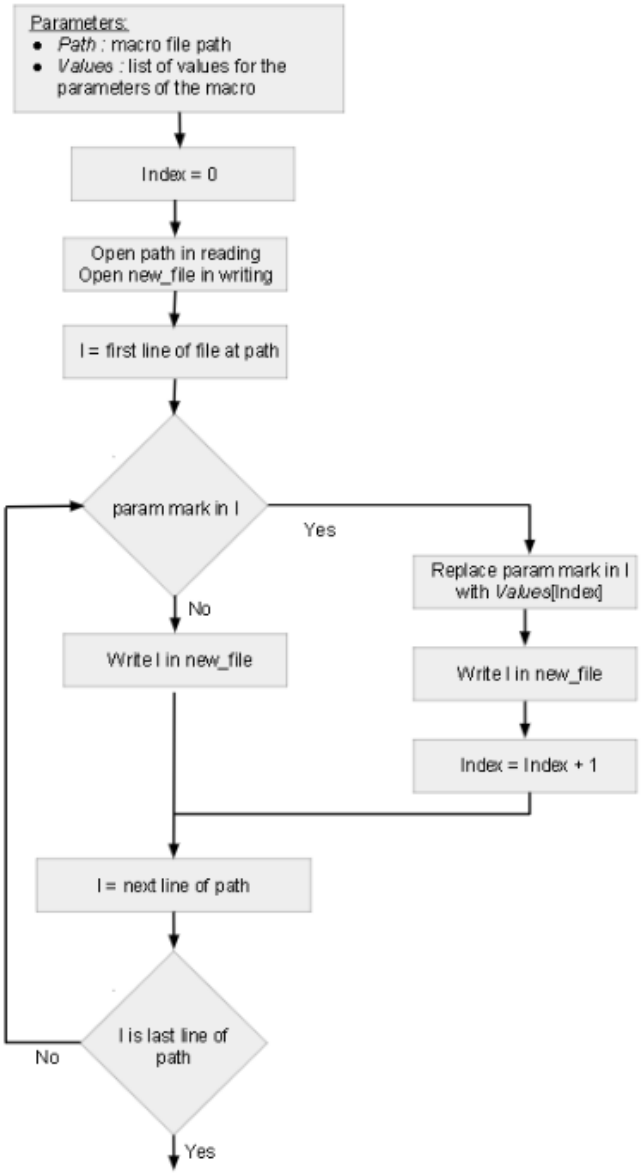


Figure 3.9: Macro alteration algorithm

For the sake of reusability the macro files are not overwritten but they are copied with the param values being replaced by the previously asked values for the parameters of the macro to be executed.

Remark:
The code is available in appendix A.2

Chapter 4

Running Examples

4.1 Basic functions

See below some needed indications for the following examples:

- The following words are keys of the one-word command hashmap: "Bonjour", "heur", "son" and "luminos"
- The following words are keys of the group-of-words command hashmap: "ça va" and "recherch internet".
- The keys are roots of words to compare with the stemmed version of the input sentence.
- The keys will be highlighted in blue in the examples.

4.1.1 One-word command in sentence

Pourrais	- tu	me	donner	l’	heure.
0	1	2	3	4	5

Table 4.1: One-word command example

Stemming is applied to each word with following result:

pourr	tu	me	don	l	heur
0	1	2	3	4	5

Table 4.2: Stemmed one-word command

Each word will be tested to know if it belongs to the keys of the one-word command hashmap. Upon the testing of the index 5 the group-of-words recognition algorithm will be launched on the words between index 0 and 5. In this case nothing will be found. And then the function associated to the key heur will be executed. In this case it will obviously announce the current time.

4.1.2 Group-of-words command in sentence

Lance	une	recherche	internet	sur	Siri.
0	1	2	3	4	5

Table 4.3: Group-of-words command example

Stemming is applied to each word with following result:

lanc	une	recherch	internet	sur	sir
0	1	2	3	4	5

Table 4.4: Stemmed one-word command

Each word will be tested to know if it belongs to the keys of the one-word command hashmap. After the testing of the whole sentence without founding anything, the group-of-words recognition algorithm will be launched on the words between index 0 and 5. In this case the key recherch internet will be found. And then the function associated to this key will be executed. In this case it will launch a Google search with "Siri" as research subject and not a joke about other IPA performances (yet).

4.1.3 Composed commands in sentence

Boujour,	ça	va?	donne	l'	heure.
0	1	2	3	4	5

Table 4.5: composed commands example

Stemming is applied to each word with following result:

bonjour	ça	va	don	l	heur
0	1	2	3	4	5

Table 4.6: Stemmed composed command

Each word will be tested to know if it belongs to the keys of the one-word command hashmap. Upon the testing of the index 0 and finding "bonjour" the group-of-words recognition algorithm will be launched on the words between index 0 and 0. In this case nothing will be found. And then the function associated to the key bonjour will be executed.

After this, the test of each word will start again. In this case it will stop at index 5. The group-of-words recognition algorithm will be launched between index 0 and 5 (if the last one was found on index 3 it would have been 3 and 5) and execute the function of the key ça va. At the end of the algorithm the function of the key heur will be executed.

As you can see the algorithm allowed multiple commands in the sentence and respected the order.

Multiple group-of-words commands issues

Since the lookup for group-of words commands is done by iterating on the keys of the group-of-words hashmap the order is not kept. This means that if the key are ordered as: "ça va" then "recherch internet" even if the words between index of the group-of-words recognition algorithm are "recherche internet ça va" the execution will be first the function of "ça va" and then the function triggered by the key "recherch internet".

This is the reason why it was mentioned earlier that a trade-off was made regarding the order for the benefit of efficiency.

Function related issues

Some functions are extracting more contextual informations from the original message. Let's take an example with two new functions affecting the volume and the brightness:

Augmente la luminosité et diminue le son.

Figure 4.1: Functions related issues example

These two functions, once triggered, will take more information from the original message. In this case the words "augmente" and "diminue" will make the functions have different behaviours. Both effects will be applied and cancel each other.

This issue could be mitigated by only taking the part of the message before the command as a contextual information but it would let less freedom to the user in his sentences. And it would not fully solve the problem so it was decided to let this issue be.

4.2 Customizable functions

4.2.1 Creation of a macro

The function ensuring the creation of a new function works with the same principles than the others functionalities. This process is triggered by the recognition of a command. The recognized key is in this case is "nouvel fonction".

Je	souhaite	ajouter	une	nouvelle	fonction.
0	1	2	3	4	5

Table 4.7: Macro creation command example

Stemming is applied to each word with following result:

Je	souhait	ajout	une	nouvel	fonction
0	1	2	3	4	5

Table 4.8: Stemmed macro creation command

Since it is a group-of-words key and there is no one-word key in the sentence there will be a run on each word before launching the group-of-words recognition algorithm and executing the function creating new macros.

There will be a request addressed to the user to fill the text he would like to change and be customizable by the word "PARAM". This point is needed to differentiate the customizable parts of the macro from others. It is the less user-friendly part of the execution but no other satisfying solution was found.

After this request the mouseRecorder software will be launched and the user will be able to register a macro with clicks and keyboards entries. At the end of the process the macro will be saved at a defined path, which is needed to be kept in memory to store the new function. For this example: the sending of an email with "PARAM" as object and content and the adress of destination set to a defined one (for example: dad). There is also a final click on the "send" button before the end of the recording.

The user will then be asked for a command to trigger this function (for the example: "mail à papa"). After verification the pronounced words will then be stemmed and stored in the corresponding hashmap based on its number of words. There it will obviously be the group-of-words hashmap with the key "mail à pap". The function that will be pointed to is generic for the execution of all the macros.

The second data structure of the macro will then be filled. To do this the names of the parameters will be asked to the user. Then, with the command as key, the path and the parameters will be added to the hashmap used for macro information (this structure was explained in subsection 3.2.2).

4.2.2 Execution of a macro

Before anything, it is important to set the context of the macro we will use for this example. Of course part of the previous creation will be used:

- **Command/key:** "mail à pap"
- **Parameters name:** "Quel est l'objet?" and "Que souhaitez-vous dire dans le mail?"
- **Functioning:** A mail is send with as destination "dad" and variable object and content.

Je	souhaite	envoyer	un	mail	à	papa.
0	1	2	3	4	5	6

Table 4.9: Macro execution command example

Stemming is applied to each word with following result:

je	souhait	envoi	un	mail	à	pap
0	1	2	3	4	5	6

Table 4.10: Stemmed macro execution command

Upon the end of the run looking for one-word commands the group-of-words recognition algorithm will be launched. Then, when testing the corresponding key, the function launching the execution of a macro will be triggered.

With the command it will gather the path of the macro file and its list of parameters leading to the following conversation to obtain the desired values:

- **IPA:** "Quel est l'objet?"
- **User:** "Repas hebdomadaire"
- **IPA:** "Que souhaitez-vous dire dans le mail"
- **User:** "Je serai là ce soir à 20h."

Then the macro file will be copied with the "PARAM" replaced with the values previously asked. Then the file will be executed. The mail will be sent to dad with "Repas hebdomadaire" as object and "Je serai là ce soir à 20h." as content.

At the next execution the macro file executed will be overwritten with the next needed macro and its needed values instead of "PARAM" marks. The prototype is of course using copies to keep the possibility to use macro functions previously defined with values still editable.

Chapter 5

Validation

5.1 Tests

Although a lot of defect prevention was done during the coding phase, some faults were still present and led to failures. At this stage of the work a validation phase was required to improve performances of the prototype by removing faults.

The high versatility of the contribution of the user provides a lot of possible variations in the input data of the software. This point in particular led to a lot of issues while testing since it was not possible to predict every customization the user would do.

This main issue led to an evolution in the way the testing was conducted. The first idea was to conduct functional testing by using category partition. However this was too limited to truly test the limitations of the prototype and other options needed to be adopted.

5.1.1 Methods

It is important to notice that structural testing was performed at the start (mainly condition and branch coverage). But since most of the functionalities were custom made and thoroughly designed and thought for optimisation the results were not significantly relevant.

Category-partition testing

In early stage each functionality was tested with functional testing (only based on specifications) to be sure the output was the desired one. To do so the category-partition testing, seen in courses [10], was chosen.

Find below the different elements for macro execution software. With first the different categories:

- **Paths:** contains parameters with string values representing a path to a file.
- **Lines:** contains parameters with string values representing lines of a file or the line supposed to replace it in the altered file.
- **Parameters:** tables containing strings representing the parameters of the macro or the wished values for these parameters.

Categories	Paths	Lines	Parameters
Choices	Null	Null	Null
	String empty	String empty	Filled table
	String length > 0	String length > 0	
Constraints	Valid string if nor empty nor null		Tables have the same size
	File extension is mrf		

Table 5.1: Macro execution category-partition testing

These tests were made using the *unittest* library of Python. The paths were leading to a test macro which was only doing a net search.

This point in particular was an issue due to the the fact that it was only using one specific macro for all the tests. To compensate for it, it was mandatory to keep the succeeding test cases and to alter the macro file or at least change the macro executed to test other environment and tasks.

Macros testing

For this part, it is important to understand how the recording works. Most of it is done through clicks, keyboard entries and environment changes. And it means that values registered (such as text or coordinates) are saved and written in the macro file in the encoding of the current environment. (The particularity was actually pointed out by a player which recorded actions for a game and found out that his parameters were not taken into account.)

Then always executing the same file was not testing the modification operated on the file itself since after the first test we knew the outcome of such a procedure for all the next ones. Even without issues in the software and procedures themselves it was important to make sure that the alteration would always be well conducted for different kind of macros.

This point led to the need to register macros using software with different encoding. Which means we obtained files encoded in utf-8 already containing an encoded part with an ISO 8859 standard encoding for example. This resulted in unreadable symbols in some case.

The path to these new files were added to the file catalogue used to test the prototype.

Beta tests

After correcting the previous issues and to poll the public opinion regarding the concept a beta test of two weeks was organised. The prototype was installed for eight persons which were not programmers. Even if the population was not that big, it allowed to gain a lot of useful information.

They are casual users of IPA using their computers daily. They answered a survey at the end of the beta period (done with Google forms). Questions can be found in appendix D.1.

The goal, beside gaining information regarding potential failures, was knowing if the concept was convincing. Because this work tries to add a new functionality and compete with already well implanted and more sophisticated IPAs, a survey seemed like the best idea to accomplish these goals.

5.1.2 Results

Category-partition testing

There was only two significant points in the results that needed a remark:

All the test cases with invalid original macro file path and parametrized macro file path failed to obtain the desired results. But when only the original path was null there was no error issued. The previously modified macro was executed because the file had not been overwritten. It has of course been modified since.

Providing tables with not enough parameters was not an issue to the program it would only change the N first "PARAM" marks. On the other hand if there was too many parameters it would simply filled all the "PARAM" marks and then execute the modified macro file with these values. This point was not altered since it is directly linked to the user recording of a macro.

Macros testing

These tests pointed out issues for the prototype since some files did not even have their "PARAM" marks recognizable because of the superposed encodings. Even worse, since we could not know which software would be used by the user then we could not know how many and which encodings would be used.

Part of the problem was due to the use of Python which does not deal accurately with such issues. The solution adopted was to use Unicode to write the new macros, while decoding to take into account all the types of end-of-line markers.

Beta tests

Only the most revealing results were picked for this subsection but all results can be found in appendix D.2. The first question was related to the most used capacity of the IPA they usually use.

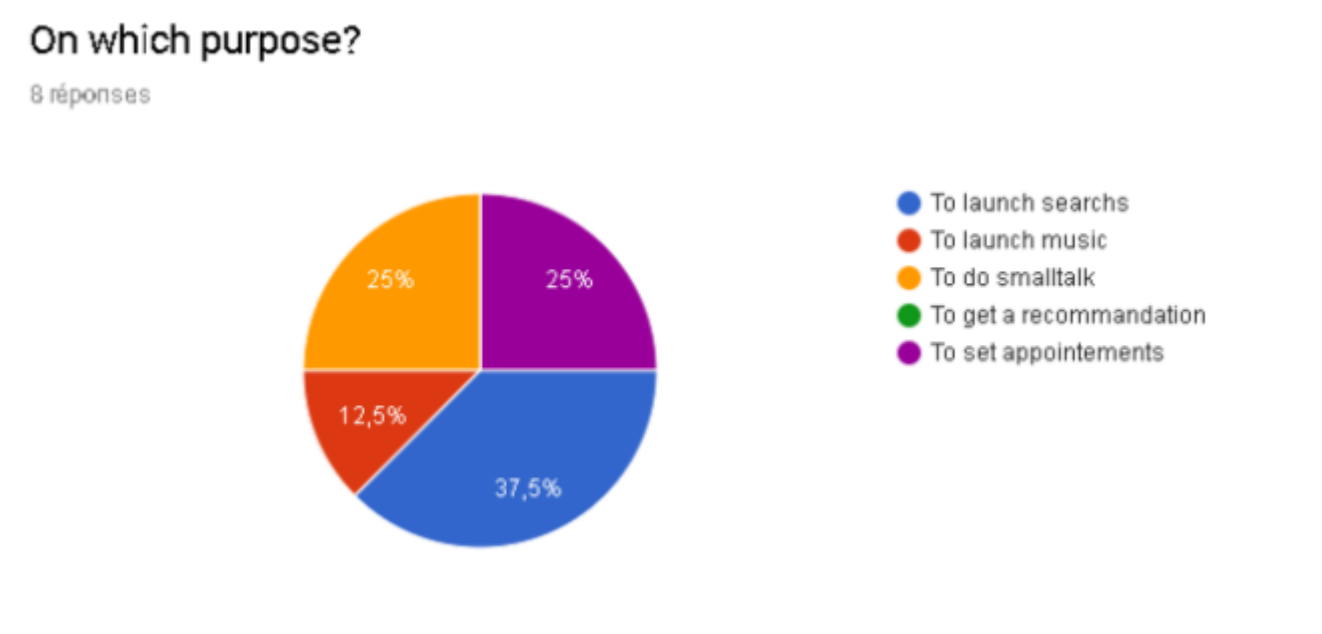


Figure 5.1: IPA uses

I thought it would be interesting to know the main functions used and expected from an IPA. Even though the search tool was of course the most used, it appears that the small talk capabilities are also appreciated. It is interesting because this particular point shows the need for a bigger database with more possible interactions.

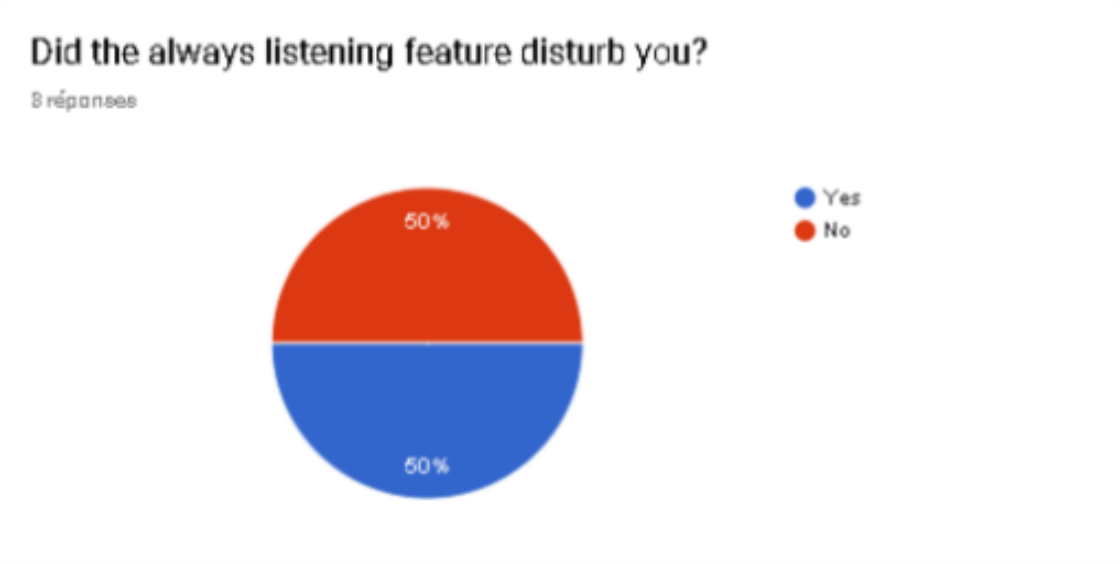


Figure 5.2: Listening feature annoyance

This graph illustrates the result of a design choice made and explained earlier in the section 3.1.4. The objective was for the user to have a more interactive relation with the IPA. And the need to call for a command word to activate the IPA was seen as an obstacle.

Results seem to indicate that it annoys half of the testers, but besides these two solutions no other solution was found.

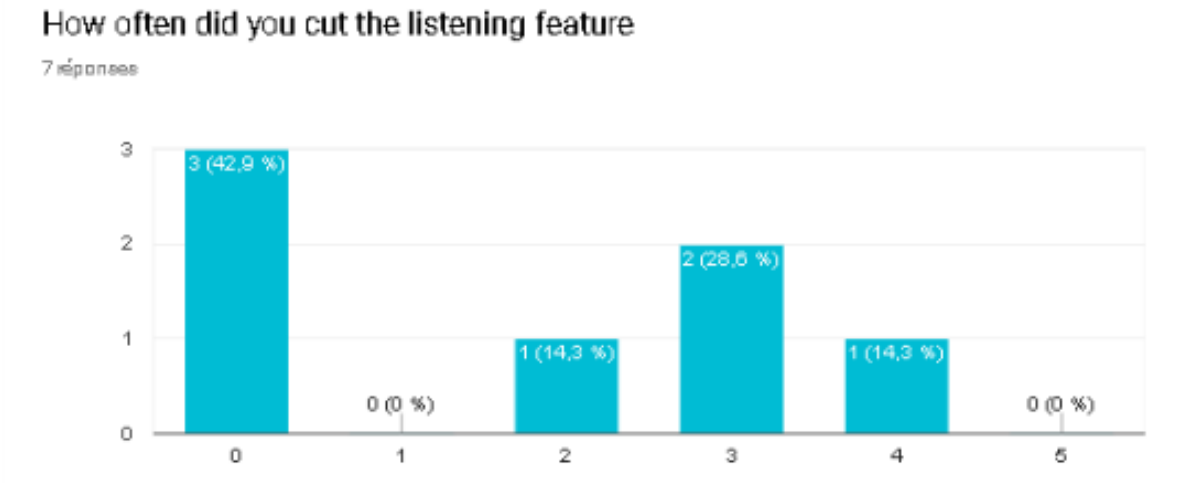


Figure 5.3: Listening feature cuts frequency

Since it is possible for the user to stop the listening feature, the previous results of course led to the concern of how often this stop happened. It seems that there are less cuts for the users privileging the small talk feature of an IPA.

The best thought alternative would be to keep the actual design choice (always listening) but improve the commands recognition with a deeper context analysis that would allow less interventions from the IPA.

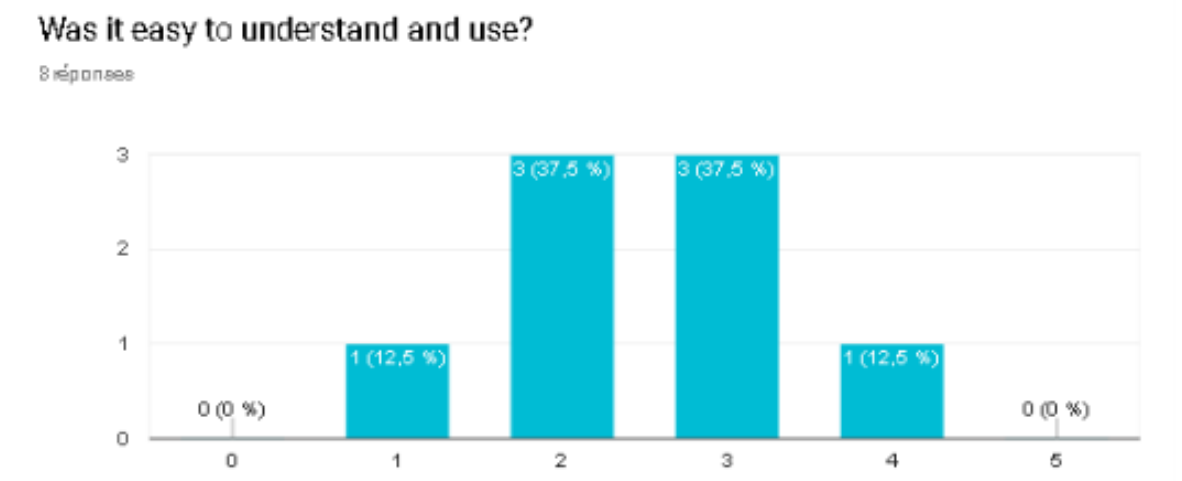


Figure 5.4: Customisation user-friendliness estimation

The testers were not programmers and were just casual computer users. The user-friendliness point in particular was important. It needed to be an achievable task for even profane user.

The results clearly shows that there is still room for improvements. The user can customize it to a large extend but it leads to let them a lot of freedom in the macro creation. An alternative could be to accompany more the user and to let him less freedom, although these restrictions

would go against the made design choices.

Although this point is arguable since it would reduce the interest of this work. But that would not deprive it from its other assets and would still be an improvement regarding the customisation from the other IPAs.

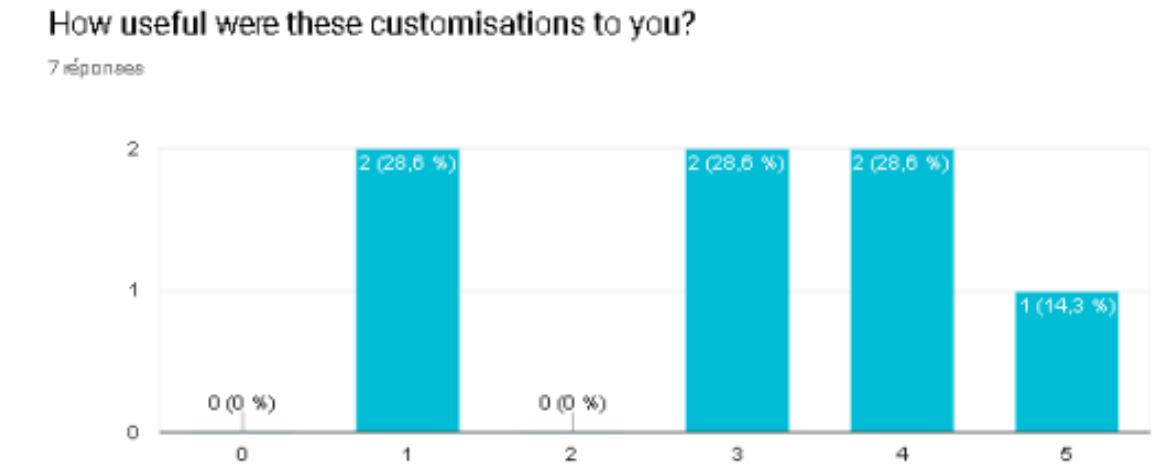


Figure 5.5: Prototype interest

According to the users who used the customization function, this option was worth it and useful to them. This point let to the next question which was focused on an economic aspect.

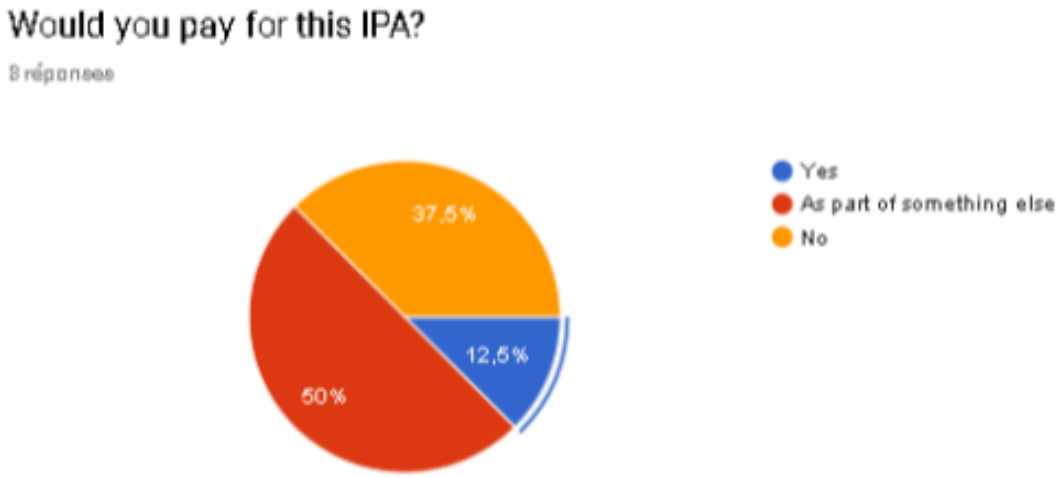


Figure 5.6: Prototype economic interest

Another question of the survey was if testers would buy the IPA if the concept was pushed to an end wand not just a prototype. This question was mainly to gauge the interest the testers had in the product.

It appears that as a stand-alone software it would not gain the approval of the users (probably due to the lack of resources and the emptiness of the databases). But as part of something else

they thought it would be a great addition. Regarding these results it may make sense to design this prototype and further develop it in a module usable with other programs.

In this line of thinking it could be an extension of an existing IPA for example. Or even a general software assistant which could be customized by the programmers with the needed capabilities specifically for the susmentioned software. (The concept was actually thought with Clippy, the word assistant, in mind.)

Chapter 6

Discussion

The whole testing process had to evolve during the validation phase because of the use of third parties software and the high impact of the user interactions. The evolution of the process was explained in the previous chapter, refer to it for more information.

Another important point is that this work is also about adding innovative abilities to an already existing technology. Another challenge was to make it reachable to casual computer users. This point may sound more like a market research but it is plenty part of this research.

It is also important to notice that no benchmark was realised to estimate the delay for performing an action or answer a question. The reason is simple: the performances are highly dependent of the internet speed since we need to reach out to online services such as Google speech recognition. The other reason lays in the language chosen to design the IPA; Python may be a great prototyping tool but it is far from being as efficient as other languages which would have make it way faster.

6.1 Limitations

Regarding the capabilities, this IPA allows a large range of functions. With the macro recorder it is possible to add nearly anything you can usually do with your mouse and keyboard.

A caution point should be the words recognized by the speech recognition service. It is why it is important to pay attention during the verification of the new commands. Some too close words from others or too complex can be misunderstood (especially if you have an accent or lisp).

6.1.1 User impact

Since they are making the recording themselves, the users have a huge impact on the execution of the software. They are the ones defining the text fields which will be customizable and later the different parameters after the recording of the macro.

Regarding that point the user could also be an issue while he specifies the values wanted for the customisation of the macro. Although he will be asked to confirm the values some problem can occur, especially when there are more than one person speaking.

6.1.2 IPA us and customs

Another issue is that the standard IPA are not that much used in our everyday life. And even with the possibility to define your own functions and add capabilities and commands to your IPA, people are just not used to rely on it.

This particular point is deserving this prototype because it requires more time to be parametrized and tuned with the preferences of the user. This is partly due to the lack of resources and the size of the programming team compared to big companies.

6.2 Improvements

Beside promoting the prototype to a big company software several ideas were thought. These points would be improvements for most of the IPAs. But there was not enough time to implement them.

6.2.1 Stream of words

By using a stream for the input of the IPA with two threads it would allow to keep listening to the user while executing functions related to what was asked earlier.

This point is already implemented in some car vocal assistants. The biggest challenge would be to keep the composed commands sentences (more than one command in a sentence) and the group-of-words commands because it would mean we could not remove too quickly words from the stream since we could need previously pronounced words. (Example: "diminuer un peu le son")

6.2.2 Portability

Make the IPA cross-platform would make it more interesting and to go further it would allow to customize the IPA and keep these customizations across devices. The issue would be that most of the macro registered for a computer would not be applicable to a smartphone for example.

6.2.3 Environment sensitivity

By adding contacts to the event and allowing the IPA to have access to these informations the prototype could be able to enquire if a meeting was well conducted. Base on this information the IPA could store informations regarding this contact and for the next meetings program reminders that the previous meetings did not went well.

Chapter 7

Conclusion

This work was the occasion to analyse the different technologies used in IPA software. It led to a clear definition of the drawbacks and advantages of these programs. This was a good starting point to define how to ameliorate and design a new IPA.

These ameliorations were mainly to provide the possibility to use more than one command in a sentence and to enable the user to add functions to its IPA. This customisation was to make it more adapted to the needs of the user.

The concept of a customizable IPA has shown its interest, but maybe in a more restricted way. The performances of the prototype depended too much on the good behaviour of the user. Allowing the user to add functionalities in a more controlled context could be a better alternative.

But there are already too many implanted software on the market which have a strong presence. Their resources and huge databases make them more reliable for an user who does not have time to set correctly the prototype with the desired commands and functionalities.

Although it was shown that it has many limitations and still needs improvements, this work led to the conclusion that the concept should come as a module to already developed IPA. Several points such as add capabilities or treating composed command sentences could be a huge update to the current top IPA, even more for a big software company since they could gather some new commands from different users.

Bibliography

[1] C. Hoffman, "Presentation of sirius," Mar 2015.

[2] T. F. for Intelligent Physical Agents, "Personal assistant specification," Oct 2001.

[3] P. Dupont and C. Fairon, *[INGI2263] Computationnal Linguistic.*

[4] Cravero, M., R. Pieraccini, and F. Raineri, *Definition and evaluation of phonetic units for speech recognition by hidden Markov models.*, vol. 11. Acoustics, Speech, and Signal Processing, IEEE International Conference on ICASSP86, 1986.

[5] V. Zwass, "Speech recognition technology," *Britannica.*

[6] "Intelligent virtual assistants," Sep 2015.

[7] "Operating system market share."

[8] S. community, "Lemmatize french text."

[9] "Speech cloud beta deprecation notice."

[10] C. Pecheur, *[LINGI2251]Software Quality Assurance.*

[11] A. Kongthon, C. Sangkeettrakarn, S. kongyoung, and C. Haruechaiaysak, "Implementing an online help desk system based on conversational agent," *National Electronics and Computer Technology Center (NECTEC).*

[12] V. de NextInpact, "L'assistant vocal cortana de windows phone 8.1 se dévoile en images," Apr 2014.

[13] S. E. Bibri, *The Shaping of Ambient Intelligence and the Internet of Things: Historico-epistemic, Socio-cultural, Politico-institutional and Eco-environmental Dimensions.* Atlantis Press, 2015.

[14] G. F. Luger, *Artificial intelligence: structures and strategies for complex problem solving.* Pearson Addison-Wesley, 2009.

[15] M. P. Peterson, *Maps and the internet.* Elsevier, 2005.

[16] D. L. Poole, A. K. Mackworth, and R. Goebel, *Computational intelligence: a logical approach.* Oxford University Press, 1998.

[17] S. J. Russell, P. Norvig, and E. Davis, *Artificial intelligence: a modern approach.* Prentice Hall, 2016.

[18] R. Sun, *Cognition and multi-agent interaction: from cognitive modeling to social simulation.* Cambridge University Press, 2009.

[19] T. Salamon, *Design of agent-based models: developing computer simulations for a better understanding of social processes.* T. Bruckner, 2011.

[20] P. Docs, “Speechrecognition - python package.”

[21] “Nircmd - freeware command-line tool for windows.”

[22] P. Docs, “Pickle: Python object serialization - python package.”

[23] “Api speech: Reconnaissance vocale - google cloud platform.”

[24] Y. Heisler, “Cortana demo fail,” Sep 2015.

[25] P. Kordík, “How deep learning and recommender systems make chatbots useful and more intelligent,” Oct 2016.

[26] Claritylab, “Lucida,” Jul 2017.

[27] M. Lutter, “Feature extraction,” Jan 2015.

[28] M. Kaelin, “Put big data to work with cortana analytics.”

[29] Tensorflow, “Useful models,” Jul 2017.

[30] Olivierverdier, “Python latex highlighting,” Feb 2017.

[31] P. Docs, “Pyttsx: Text-to-speech x-platform - python package.”

[32] W. community, “Speech synthesis,” Aug 2017.

[33] W. community, “Virtual assistant (artificial intelligence),” Jul 2017.

Appendices

Appendix A

Algorithms

A.1 Command recognition

```
words = re.findall(r"[\w]+", msg, re.UNICODE)
last_index = 0

for idx in range(len(words)):
  w = stemmer.stem(words[idx])

  if w in trigger_ht.keys():
    for k in composed_trigger_ht.keys():
      if k in u" ".join(w for w in words[last_index+1:idx]):
        composed_trigger_ht[k](msg)
    last_index = idx

    trigger_ht[w](msg)

for k in composed_trigger_ht.keys():
  if k in u" ".join(w  for w in words[last_index+1:]):
    composed_trigger_ht[k](msg)
```

Figure A.1: Command recognition algorithm

A.2 Macro alteration

```
i = 0
with open(file_path, 'U') as macro_model, open('Macros\customedMacro.mrf','U
                                        ') as customed_macro:
  for line in macro_model:
    if 'P\x00A\x00R\x00A\x00M' in line:
      customed_macro.write(custom(line, param_list[i]))
      i+=1
    else:
      customed_macro.write(line)
```

Figure A.2: Macro alteration algorithm

Appendix B

Source code

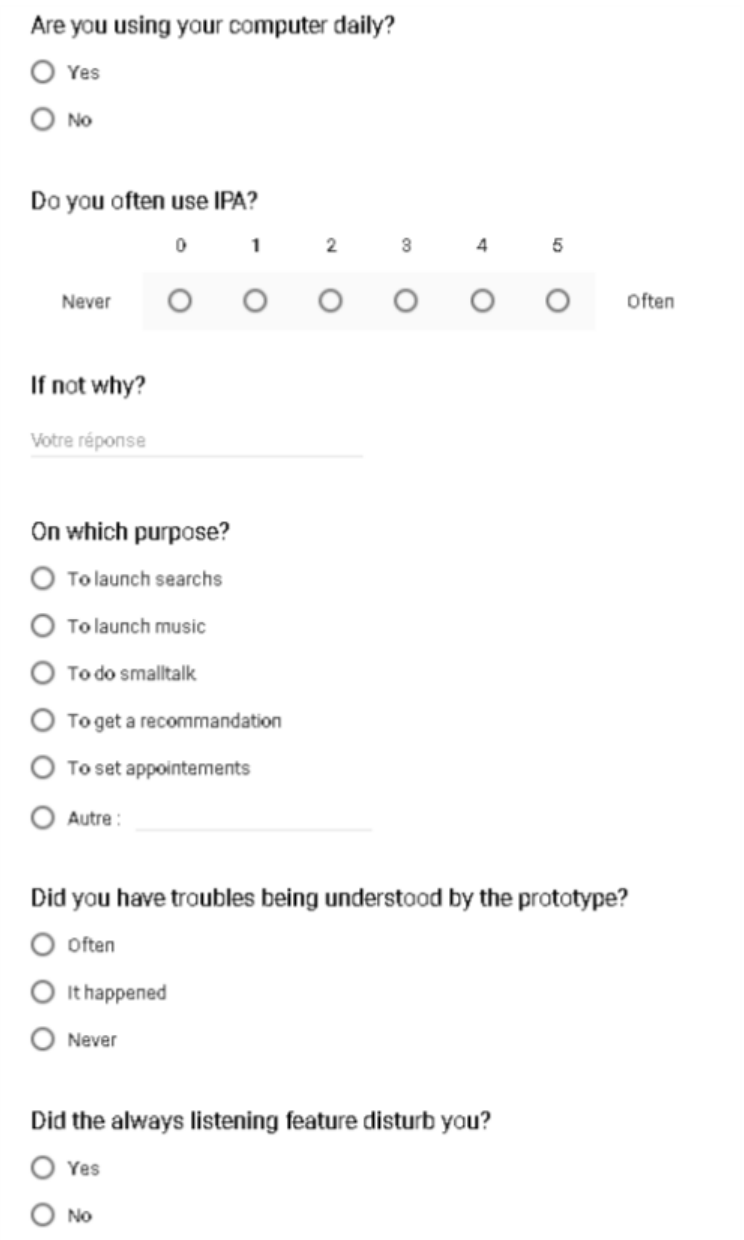
Most of the source code of the prototype (All the public functionalities not using accounts or passwords) is available together with the associated tools that have been used to develop this software on the GitHub repository of the project: `https://github.com/tdba/Customizable-vocal-personal-assistant-thesis.git`.

Note however that because another (private) repository was used most of the time, the GitHub commit log does not properly reflect the progression of the development.

Appendix C

Beta test survey

C.1 Questions



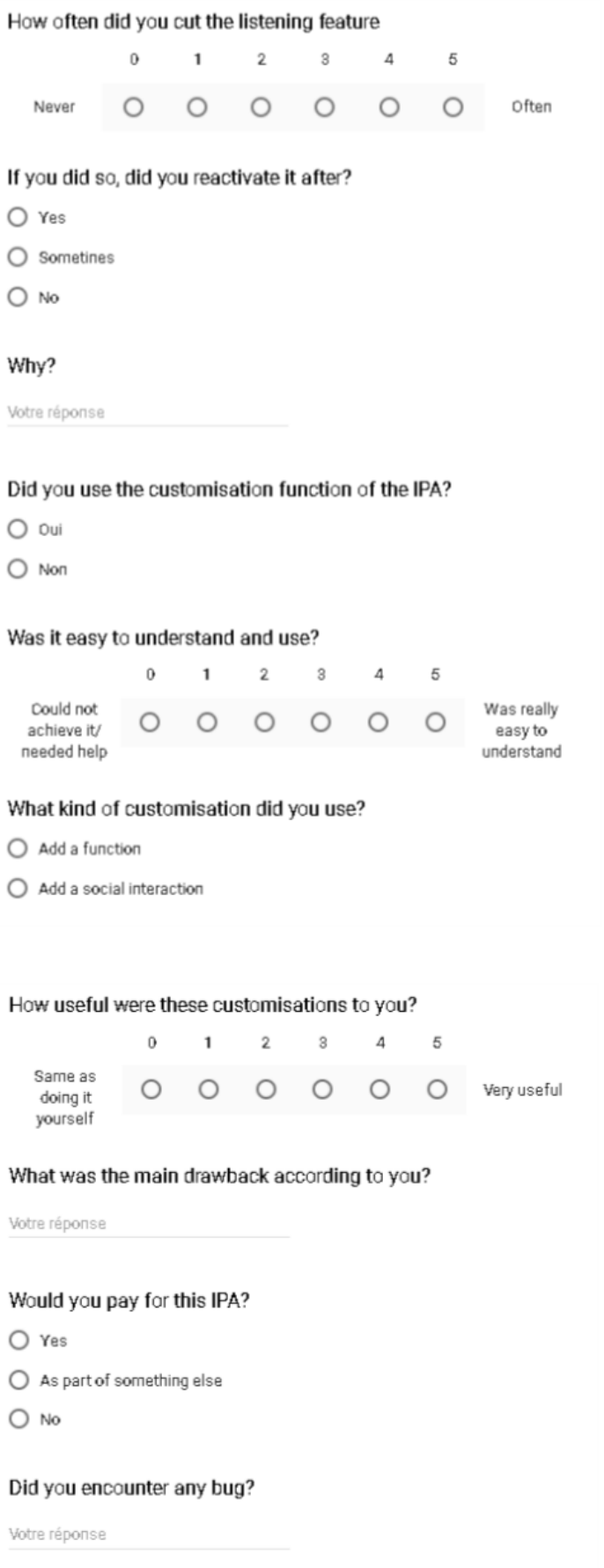
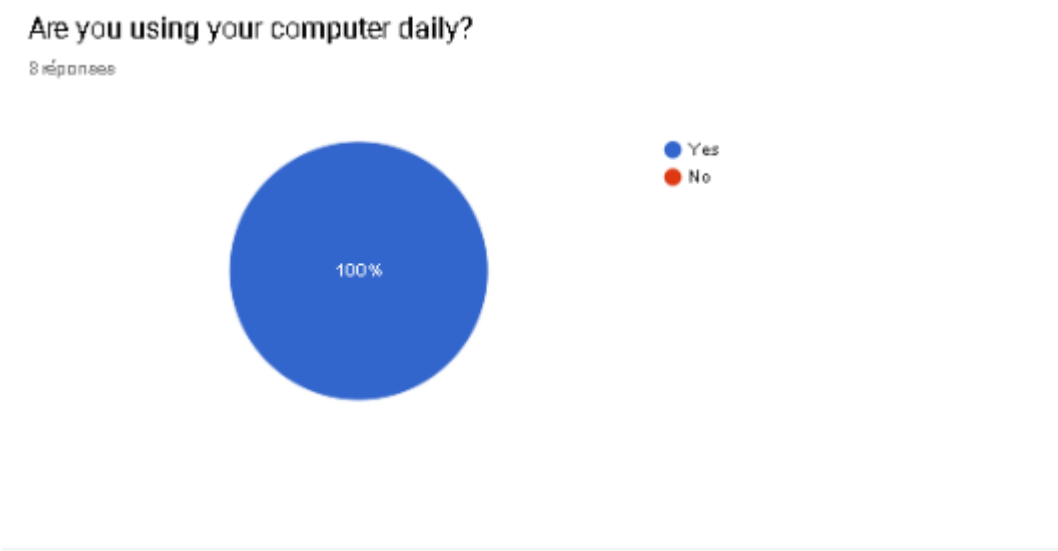
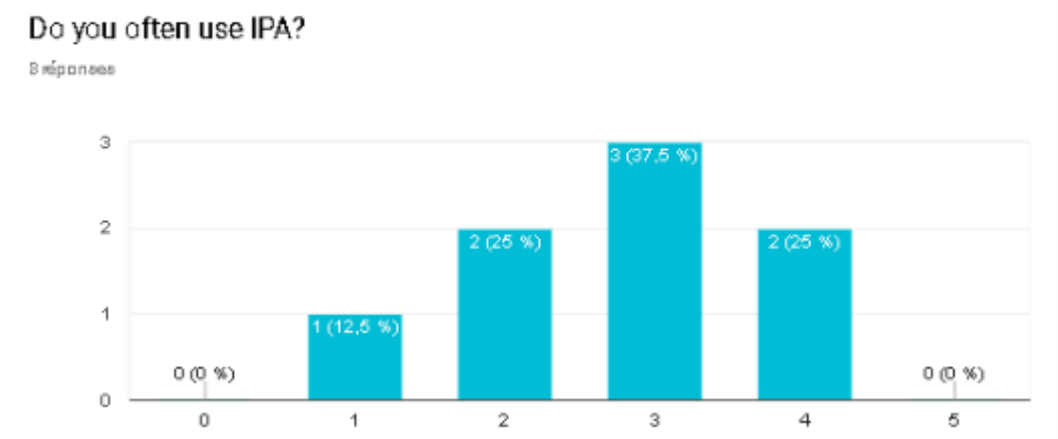


Figure C.1: Survey questions

C.2 Results


Are you using your computer daily?
8 réponses
Yes
No
100%


Do you often use IPA?
8 réponses
3
2
1
0
0 (0 %)
1 (12,5 %)
2 (25 %)
3 (37,5 %)
2 (25 %)
0 (0 %)
0
1
2
3
4
5

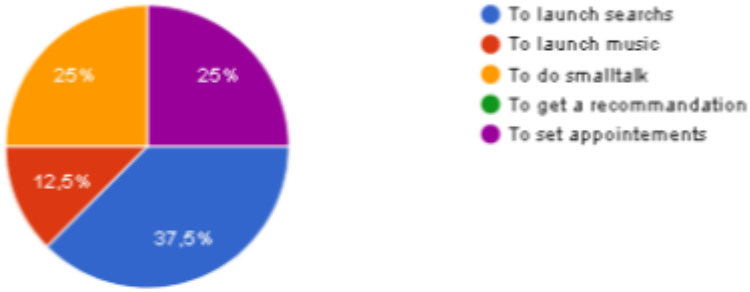
If not why?

Une réponse

Faster to do it myself than think about the right words and hope it will understand

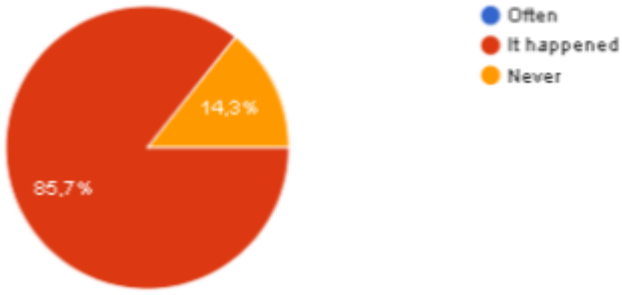
On which purpose?

8 réponses



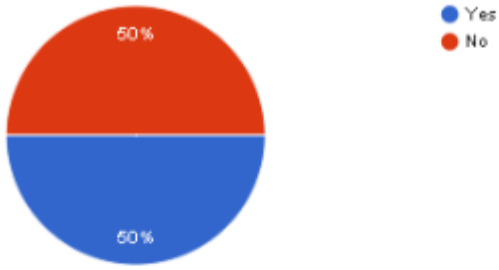
Did you have troubles being understood by the prototype?

7 réponses



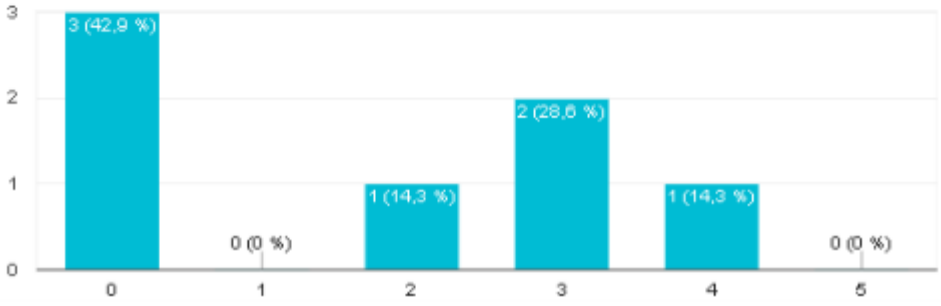
Did the always listening feature disturb you?

8 réponses



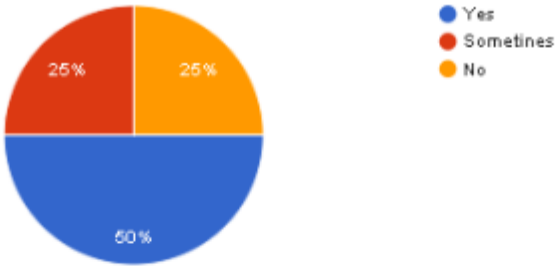
How often did you cut the listening feature

7 réponses



If you did so, did you reactivate it after?

4 réponses



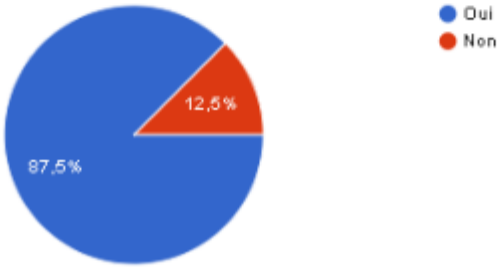
Why?

2 réponses

Quand j'en avais besoin
Forgot

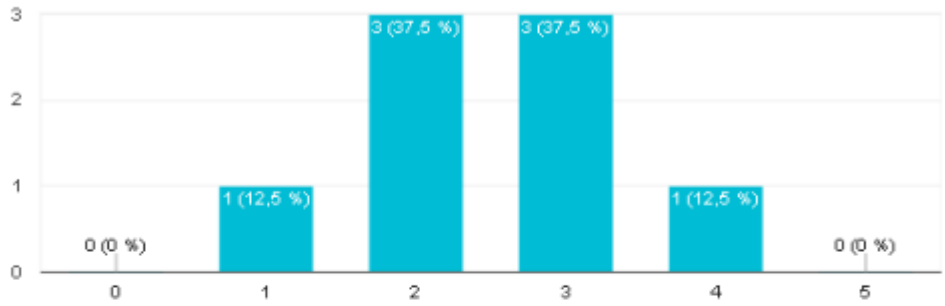
Did you use the customisation function of the IPA?

8 réponses



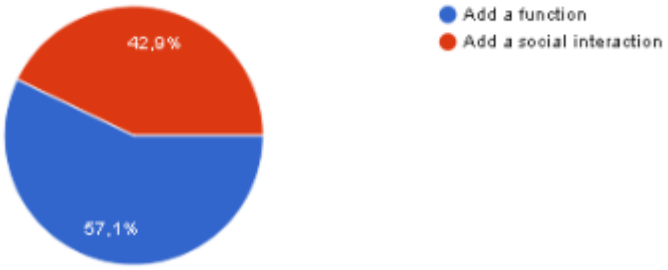
Was it easy to understand and use?

8 réponses



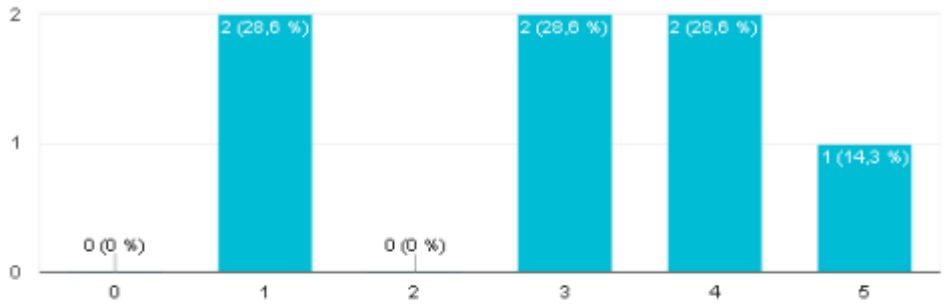
What kind of customisation did you use?

7 réponses



How useful were these customisations to you?

7 réponses



What was the main drawback according to you?

8 réponses

Quand je parlais avec des gens il réagissait alors que je n'avais pas besoin de ca
Took me a lot of time to add all the wanted functions
Was always reacting to me before I set it to not do it
The basic answers were quite poor
M'écoutant tout le temps, le programme réagissait même quand je ne voulais pas lui parler
Did not see the interest of the customisation tool, basic IPA are effective enough
It didn't always understand what I said and it was annoying regarding searchs
The customisation is not intuitive enough

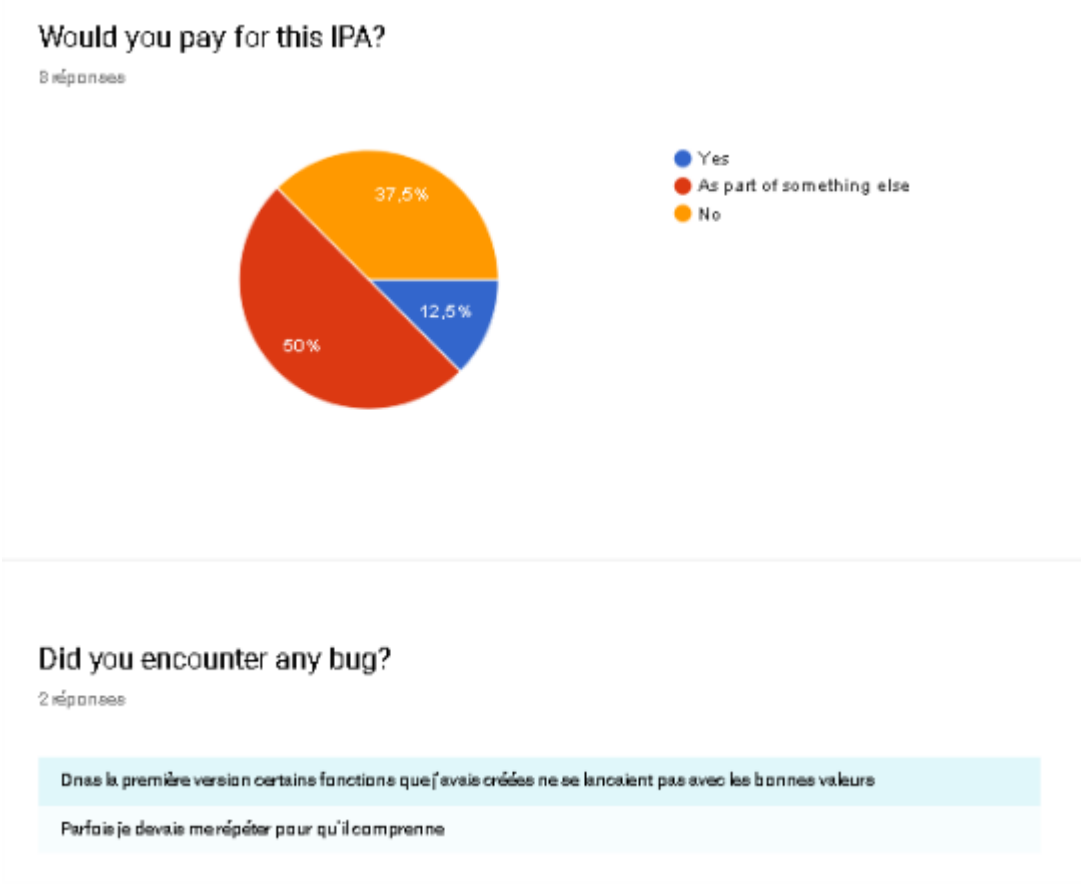


Figure C.2: Survey answers

Rue Archimède, 1 bte L6.11.01, 1348 Louvain-la-Neuve **www.uclouvain.be/epl**