AKADEMIA GÓRNICZO-HUTNICZA IM. STANISŁAWA STASZICA W KRAKOWIE

WYDZIAŁ INFORMATYKI, ELEKTRONIKI I TELEKOMUNIKACJI

KATEDRA Telekomunikacji

PROJEKT INŻYNIERSKI

Voice interface for telephone usage of Internet shop

Głosowy interfejs do obsługi telefonicznej sklepu internetowego

Autor: *Marcin Styczeń*
Kierunek studiów: Elektronika i Telekomunikacja w języku angielskim
Opiekun pracy: *dr. inż. Bartosz Ziółko*

Kraków, 2016

Uprzedzony o odpowiedzialności karnej na podstawie art. 115 ust. 1 i 2 ustawy z dnia 4 lutego 1994 r. o prawie autorskim i prawach pokrewnych (t.j. Dz.U. z 2006 r. Nr 90, poz. 631 z późn. zm.): „ Kto przywłaszcza sobie autorstwo albo wprowadza w błąd co do autorstwa całości lub części cudzego utworu albo artystycznego wykonania, podlega grzywnie, karze ograniczenia wolności albo pozbawienia wolności do lat 3. Tej samej karze podlega, kto rozpowszechnia bez podania nazwiska lub pseudonimu twórcy cudzy utwór
w wersji oryginalnej albo w postaci opracowania, artystyczne wykonanie albo publicznie zniekształca taki utwór, artystyczne wykonanie, fonogram, wideogram lub nadanie.", a także uprzedzony o odpowiedzialności dyscyplinarnej na podstawie art. 211 ust. 1 ustawy z dnia
27 lipca 2005 r. Prawo o szkolnictwie wyższym (t.j. Dz. U. z 2012 r. poz. 572, z późn. zm.) „Za naruszenie przepisów obowiązujących w uczelni oraz za czyny uchybiające godności studenta student ponosi odpowiedzialność dyscyplinarną przed komisją dyscyplinarną albo przed sądem koleżeńskim samorządu studenckiego, zwanym dalej „sądem koleżeńskim", oświadczam, że niniejszą pracę dyplomową wykonałem(-am) osobiście, samodzielnie i że nie korzystałem(-am) ze źródeł innych niż wymienione w pracy.

List of contents

1. Introduction

1.1 General description

Main goal of this document is to describe whole exemplary process of creating demo version of voice interface for internet shop. Nowadays very popular concept of simplifying customers serving and improve enterprise functioning. Every telecommunications providers, shops, medium and big companies and many others use voice interfaces. There is one of many crucial benefits. Hiring real person is not necessary. It is very important from financial point of view. Also it is possible to serve many customers in the same time. Queue problem – solved. The main defect is as follows: this low level AI is not infallible. It is rather recommended to avoid situation with mistaken order or registering wrong personal data. Those kind of mistakes could cause deterioration of company repute. There are much more advantages and disadvantages of such solution. Anyway it is easy to observe everything moves towards to automatization.

At the beginning one thing should be explained. Whole document is written in English but speech pattern is in polish language. This is caused by SARMATA which is polish recognizer system. Applying English words could introduced many problems. Every polish sentences will have exact translation.

This topic covers mainly two domains : programming , voice processing. Whole aspects introduced here will be described more detailed in later chapters. There were many ways how to cope with this task. The most proper one (for this level of study) was to use some already created speech processor and based on this add some code to perform particular tasks. Whole project is written in IDE(integrated development environment) MS Visual Studio which consists all necessary tools (e.g. debugger, compiler) in one software. Polish speech system recognition SARMATA was chosen developed on Department of Electronics on AGH. Thanks to stability and reliability of this system whole project was created with little effort. Next necessary thing is to create grammar file. It consists the most likely words which will be pronounced by client. Created in *ABNF*(Augmented BNF)[1]. A language in which we are able to create some instructions how and what our speech recognizer should interpret speech commands. Significantly simplify comprehension of whole conversation process. Depending on task complexity many grammar files can be added to maintain system readable. Next problem was to decide how to generate

[1] *Speech Recognition Grammar Specification Version 1.0* Dec. 2015, ULR: http://www.w3.org/TR/speech-grammar/

automat voice. There were few options like record them using microphone or to generate them by software. Finally IVONA was used to generate them. For establishing connection *Voip*[2] technology is used. The person who is interested in order some products from shop have to call particular *Voip* number. In fact while application is running customer will heard voice played from memory.

Following chapters will explain abovementioned aspects more precisely.

[2] *Voice over IP* Dec. 2015 ,URL: https://en.wikipedia.org/wiki/Voice_over_IP

1.2 Historical review

Speech processing is the issue which applications can be found in many filed of human existence. Starting from early to nowadays researches people always wanted to interact with machine using human voice. Implementation of speech processing significantly contributed on human life making it easier and safer.

Everything has began in 1932 in Bell Labs when Harvey Fletcher[3] started to think how we can make machine to be able to understand what people say. World has to wait to 1952 to see first results. In the same lab "Audrey" was created. Very primitive system where only one digit commands were understandable for machine. Whole process of recognition based on locating formants in the power spectrum. 50's was the era when systems were restricted to single-speaker and around ten word vocabulary. Ten years later IBM shown "Shoebox" which dictionary were extended to 16 English words[4]. Main problem of developing speech processing was associated with low computational power. So improvements like support four vowels and nine consonants were great successes. In the late 1960's Raj Raddy solved another problem. Reddy's system could understand continuous speech. It was created for chess game. In the same time in Soviet Russia dynamic time warping algorithm was prepared which made machine capable to operate on 200-words[5]. Events like developing of Markov chains and find usage by James Barker and Janet Barker for Hidden Markov Model[6] for speech processing were also important. These research gave ability to simplify understanding acoustics, language and syntax in the form of combining them to one unified statistical model. In the 1970 DARPA[7] has sparked a project which predictions were to create system with minimum 1000 words in dictionary. The effect was "Harpy" system , 1011 understandable words. DARPA speech processing project was not successful at all because of many unreached goals nevertheless crucial improvements have taken place like

[3] *Speech recognition* Dec. 2015, URL: https://en.wikipedia.org/wiki/Speech_recognition

[4] Christoper Schmandt , ***Voice Communication with Computers: Conversational Systems*** , 1993, Van Nostrand Reinhold , 0442239351

[5] Juang B.H, Rabiner Lawrenece R. *Automatic Speech Recognition - A Brief History of the Technology Development* , 2004, Dec. 2015, URL: http://www.idi.ntnu.no/~gamback/teaching/TDT4275/literature/juang_rabiner04.pdf

[6] *Hidden Markov Model* Dec. 2015, URL: https://en.wikipedia.org/wiki/Hidden_Markov_model

[7] *Speech Recognition Through the Decades: How We Ended Up With Siri* Dec. 2015, URL: http://www.pcworld.com/article/243060/speech_recognition_through_the_decades_how_we_ended_up_with_siri.html

invention of beam search , linear predictive coding and cepstral analysis[8]. What is more first commercial speech recognition companies were created like Threshold Technology. Researches from 70's also gave us ability to create system which could interpret multiple people's voices. Progress from 80's allowed to reach several thousand words vocabulary with predictions to recognize unlimited number of words[9]. Thank to IBM research under Jack Jelinek lead "Tangora" could handle 20,000 words dictionary. They put a lot effort to develop new view on speech processing. By using statistical modeling techniques they tried to simulate how in fact human brain processes speech. Again , HMM (mention above) was crucial in Tangora research which become the most influential algorithm in 1980's in speech recognition research.

In 1990's we had contact with first speech recognition software designed for ordinary people usage. Thanks to increased computational power Dragon Dictate product appeared. It costed quite lot - $9000. But it gave fundamentals to create Dragon Naturally Speaking. "Naturally Speaking" by the fact that this system was able to understand continuous speech. With efficiency 100 words per minute and requirement to train the program some about 45 minutes we were approaching with small steps to nowadays complex speech recognition systems.(Still expensive - $695). The progress was proven by the fact that system vocabulary was bigger than average human vocabulary. At the beginning of 21th century two very important programs were sparked: Effective Affordable Reusable Speech-to-Text (EARS) in 2002 and Global Autonomous Language Exploitation (GALE). The first one was developed by IBM, BBN and Cambridge University. In 2006 National Security Agency created technology called keyword spotting. This solution enabled us to search particular word or conversation in large volumes. Data of interest could be indexed and stored in database to further analysis. New techniques like deep learning[10] was introduced. It figured out that deep learning could decrease word error rate by 30%. The solution was adopted quickly. And then another big player came on the market. Google who created GOOG-411 in 2007 in some years later developed their system to Google search voice which supports 30 languages. This corporation put big effort to improve speech processing. Starting form origin when systems could incorporate with 10 to 100 words. Nowadays this number increased to

[8] *Cepstral Analysis of Speech* Dec. 2015, URL: http://iitg.vlab.co.in/?sub=59&brch=164&sim=615&cnt=1

[9] Mihelic France , Zibert Janez ,*Speech Recognition* , 2008, InTech , 9789537619299

[10] *Deep learning* Dec. 2015, URL: https://en.wikipedia.org/wiki/Deep_learning

230 billion words. Voice Search product which uses this big word database can be found in android OS or in chrome browser to simplify human life. It is also have be mentioned about Siri which is the response from Apple for Google solutions. This is intelligent software assistant for iOS users. Microsoft also created this kind of assistant and it is called Cortana. Using those tools we have impression that we have conversation with real person. "knowing everything" assistants thanks to easy access to internet. The future belongs to systems which would be able to control our device in more efficient way. Improve the noise immunity is the next issue.

1.3 Shop description

To perform whole task shop and list of products is required. Taking into consideration that it is only demo imaginary shop called Greengrocer have been created. Client is able to buy vegetables and fruits with delivery option. To simplify task it is assumed that all products are countable.

1.4 Voip technology[11]

In modern world exists many ways how to establish telephone call. It is noticeable that people start to use internet calling more frequently. One of the way is to use Voip.

- *Voice over IP*: technology which allows us to transmit voice or other multimedia sessions using IP protocol. It is an alternative for ordinary telephone , PSTN(public switched telephone network). The greatest Voip advantage is that client pay only for transmitted data. System detects silence , suspends connection and listens to caller voice.

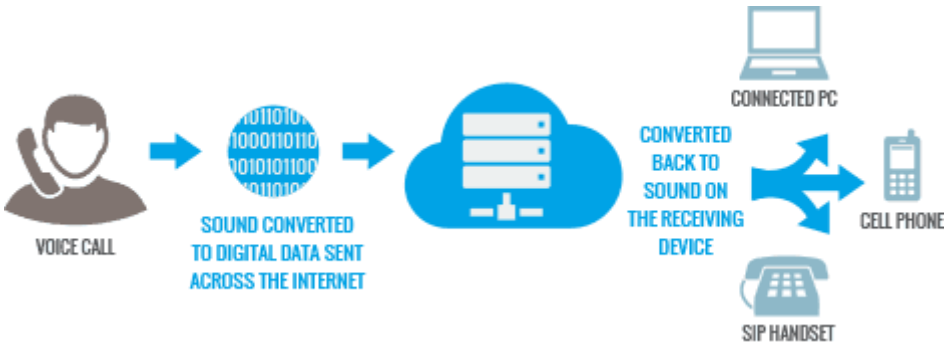

Figure 1 Voip operation

[11] *Voice over IP* Dec. 2015 ,URL: https://en.wikipedia.org/wiki/Voice_over_IP

It is necessary to obtain number under which we can establish call. *https://www.happycall.pl/*[12] have been used to create number **48123454639**. This number is the number where client can call to Greengrocer.

[12] *Voip access* Dec. 2015, URL: https://www.happycall.pl/

2. Experiment

2.1 Voice samples preparation

All of the samples were prepared by widely available https://www.ivona.com/pl/ website. They were recorder by Windows Recorder. All of them are stored in *pregenerated_tts* folder.

2.2 Configuration files

IVR application requires special configuration files. One of these is the file where everything associated with connections has to be defined. "TaxiIvr" contains such information.

```
[asr]
host=149.156.121.193
port=1414
```

Figure 2 ASR configuration

First of all information about ASR have be to defined. *host* is the address IP where software must refer to begin process of recognition under abovementioned *port* number 1414.

Now SIP[13](session information protocol) has to be configured. This protocol is mainly used to control various communication sessions. SIP is proper for IVR applications.

```
[sip.account]
domain=sip.2call.pl
registrar=sip.2call.pl
user=48123454639
password=Ykmd6y2y

[sip.settings]
transport_port=5060
```

Figure 3 Sip configuration

domain and *registrar* shows Voip provider server. *user* and *password* fields gives information about Voip account. Connection is established on port 5060.

[13] *Session Initiation Protocol* Dec. 2015, URL: https://en.wikipedia.org/wiki/Session_Initiation_Protocol

2.3 Grammar files

It was mentioned some information about grammar files which are crucial for create reliable and fast speech recognition systems. Software just take necessary information from these files.

In grammar file words or even whole sequence of words on which speech recognizer should listen to are specified. There are two main languages for cope with this task: ABNF and XML. It is possible to convert one to another.

- *Augmented Backus-Naur Form syntax(ABNF)*[14]*:* plain-text representation , widely used in speech recognition field. It ensures service for bidirectional communications protocol.

To processes grammar file *grammar processor* and *user agent* is needed. The first one is for accepting grammar file as input. User agent takes user input and matches those data with words in grammar file to produce a recognition result. Data type accepted by user agent depends on modes of grammar. For instance: speech input for voice or for DTMF(Dual-Tone-Multiple-Frequency). Grammar file points out what recognizer should listen for so: words that may by spoken, pattern where we can notify occurrences of given words. It is recommended to create few grammar files. Each of them is responsible for some part of whole vocabulary. This manipulation leads to create more readable and understandable system. System of many grammar files creates whole tree. *grammars* folder consists all of them used in project. Each file corresponds to particular part of dialog.

```
#ABNF 1.0;

language pl;

mode voice;

root $product_grammar;

tag-format <semantics/1.0>;

$product =  $cebula | $pomidor | $pomarańcza | $jabłko | $seler | $marchew | $papryka | $sałata | $banan | $cykoria |
$ogórek | $por | $śliwka | $czosnek | $ananas | $kapusta | $kokos | $mango | $arbuz | $brzoskwinia | $truskawka |
$kukurydza | $burak | $chrzan | $melon | $gruszka | $brokół | $czosnek | $dynia | $rzodkiewka | $ananas | $grejpfrut |
$morela | $poziomka | $ziemniak;
```

Figure 4 Product grammar file

In *product_grammar* all available product are included. Either vegetables and fruits. First line informs that ABNF version 1.0 is used. Next line determine used language. It is voice interface so mode is voice.

[14] *Augmented Backus - Naur Form* Dec. 2015, URL: https://en.wikipedia.org/wiki/Augmented_Backus%E2%80%93Naur_Form

```
tag-format <semantics/1.0>;
```

Figure 5 tag-format configuration

This is a *tag-format* declaration. Next is the *tag format identifier*[15].

ammount_grammar contains numerals for determine how many products client want to buy. This file is quite large and inserting it here could make little mess. For demo purposes restriction from 0 to 100 numbers was used.

```
#ABNF 1.0;
language pl;
mode voice;
root $decision_grammar;
tag-format <semantics/1.0>;
$decision = tak | nie;
```

Figure 6 Decision grammar file

decision_grammar has only *$decision* with options 'tak' or 'nie' (yes or no). Rules are created to indicate that in given point in grammar file some specific words will be spoken. It will be necessary to decide whether client want buy something more or not.

street_gramamar describes all streets where package can be delivered. For demo purposes limitation for Cracow Krowodrza district streets was made.

2.4 Code

SARMATA speech recognition system is software. It is the set of many source files which cooperate with others to create one big computer program. By some code manipulation we can set it to perform determined task. This section describes "brain" of system. The most complicated with software intelligence. Some already created code like SIP[16] protocol configuration is reused for this projects purposes.

SARMATA is written in C++ language[17]. Quite simple , reliable, objective computer language. Code is composed in Microsoft Visual Studio.

In file *IvrDialogLogic.cpp* whole conversation logic is described.

---

[15] *Speech Recognition Grammar Specification Version 1.0* Dec. 2015, ULR: http://www.w3.org/TR/speech-grammar/

[16] *Session Initiation Protocol* Dec. 2015, URL: https://en.wikipedia.org/wiki/Session_Initiation_Protocol

[17] *C++ Documentation* Dec. 2015, URL: http://www.cplusplus.com/

```
#include "IVRDialogLogic.h"
#include <regex>
#include <chrono>
#include <random>
#include <boost/filesystem.hpp>
#include <boost/filesystem/fstream.hpp>
#include <boost/algorithm/string.hpp>
#include "SIPEngine.h"
#include "ASREngine.h"
#include "ASRSession.h"
#include "SynchronizedOutput.h"
#include "StrUtils.h"
#include <sstream>
```

Figure 7 Necessary libraries

Taking into account complexity of code a lot of libraries should be included. First *"IVRDialogLogic.h"* corresponds to header file where description of main class *IVRDialogLogic* can be found. This is a typical form of composing computer code to make code more readable. *regex* , *chrono, random*[18] are standard C++ libraries. The first on is for regular expressions. It is helpful when we want to find out how our input match to some predefined pattern. Next includes are associate with boost library. This library give us big support for work with structures. Covers multithreading, regular expression. In *SIPEngine.h* exists declaration of Session Initiation Protocol. Necessary solution for establishing calls. *ASREngine.h* - managing speech recognition with SARMATA server. *ASRSession.h* - file for maintain session with Automatic Speech Recognition. *STRUtils.h* - to make out output streams synchronized. *ssteram* is the library for type conversion.

```
const size_t IVRDialogLogic::MAX_RECOGNITION_FAILURES = 3;

IVRDialogLogic::IVRDialogLogic(std::shared_ptr<SIPSession> &sipSession,
    TTSEngine &ttsEngine,
    ASREngine &asrEngine,
    SIPEngine &sipEngine)
    : IVRSession(sipSession, ttsEngine, asrEngine, sipEngine)
    , m_recognitionFailures(0)
{
    SynchronizedCout.SafeUse([](std::ostream &output)
    {
        output << "IVRDialogLogic::IVRDialogLogic -- object successfully created" << std::endl;
    });
}

IVRDialogLogic::~IVRDialogLogic()
{
    SynchronizedCout.SafeUse([](std::ostream &output)
    {
        output << "IVRDialogLogic::~IVRDialogLogic -- object successfully destroyed" << std::endl;
    });
```

Figure 8 Establishing ASR and SIP engine

[18] C++ *Documentation* Dec. 2015, URL: http://www.cplusplus.com/

This part of code basically give us information in prompt that connection is established.

```
*** PJSUA2 STARTED ***
IVRSession::IVRSession -- object successfully created
IVRDialogLogic::IVRDialogLogic -- object successfully created
```

Figure 9 Prompt after successful connection

more precisely that objects which are responsible for making connections are created properly

```
IVRSession::OnSessionTerminated
terminate called more than once
IVRDialogLogic::~IVRDialogLogic -- object successfully destroyed
IVRSession::~IVRSession -- object successfully destroyed
InputAudioStream::~InputAudioStream -- destructor sucessfully finished
OutputAudioStream::~OutputAudioStream -- destructor sucessfully finished
```

Figure 10 Terminating program

and successfully destroyed.

Later *Say()* appeared which is the *IVRDialogLogic* method that will produce on prompt the name of currently playing sample.

```
IVR says: :: Witaj w sklepie ::
```

Figure 11 IVR welcome

This method takes as argument input text and then produce it on the prompt. This say is not required. It acts like additional information for administrator that this or that message is produced to customer. *Say()* has also another great property. Signal transfer is handled by thread. It is necessary to possess function which task will be to recognize what kind of fruit or vegetable are we talking about in given moment. This task will be done by *Recognize()* method. This method is a *Recognition* class type. As first input argument this method takes vector of strings which in fact is what we say and second is the proper grammar file. The next very important method is *Run()*. Its task is to check if possible and run SIP session to establish connection. Now finally we can focus on main core of conversation logic. This code is inside *Dialogue()* method.

```
bool dec = true;
std::ostringstream strs;
std::vector<OrderInfo> Order;

std::string street_grammar = "street_grammar";
std::string product_grammar = "product_grammar";
std::string amount_grammar = "amount_grammar";
std::string decision_grammar = "decision_grammar";
```

Figure 12 Declaring necessary elements to create dialog

dec Boolean type is for determine condition in while loop whether client wants order something more or not. Second declaration is made to create later conversion from double to string. In next line vector *Order* of *OrderInfo* type structure is created. Next four string are responsible for point out grammar files to software. [19]

```
do{

    OrderInfo oi;                                   // instance of OrderInfo structure creation
    auto product = Recognize("Produkt",product_grammar);  // product recognition
    oi.product = product.text;                      // call product field in oi instance
    Order.push_back(oi);                            // insert information to vector

    auto amount = Recognize("Ilosc", amount_grammar);
    oi.amount = amount.text;
    Order.push_back(oi);

    auto decision1 = Recognize("Decyzja",decision_grammar);
    oi.decision1 = decision1.text;
    Order.push_back(oi);

    if(oi.decision1=="nie"){                            // check if client want byu something more
        dec=false;                                  // if not dec is false
    }

}while(dec);                                        // so while loop terminates
```

Figure 13 Order acceptance section

Then acceptance of order performs. By the glance on *OrderInfo* structure:

```
struct OrderInfo {
    std::string product;
    std::string amount;
    std::string decision1;
};
```

Figure 14 OrderInfo structure

It is easy to observe what fields consists *oi* instance. After recognition process and conversion to string data is inserted to the vector. The same pattern occurs for determine amount(Ilość) and decision(Decyzja). If client say "tak"(yes) new instance *oi* is created for next order.

[19] Stroustrup Bjarne *The C++ Programming Language , 4th Edition* , 2013 , Addison-Wesley, 9780321563842

```
double total_cost = 0;
for (auto& j : Order) {
    total_cost += prices[j.product];          // total cost of ordered products
}                                             // is computed
```

Figure 15 Calculating total cost section

Thanks to simplification introduced in newer C++ standards it is quite easy to compute total amount of money which client have to pay. *total_cost* variable stores this value. It is done by extraction of price data from *prices* map.[20]

```
std::map<std::string, double> prices;
prices["pomidor"] = 0.30;
prices["cebula"] = 0.05;
prices["pomarańcza"] = 0.60;
prices["jabłko"] = 0.20;
prices["seler"] = 0.43;
prices["marchew"] = 0.30;
prices["papryka"] = 0.21;
prices["sałata"] = 0.40;
prices["banan"] = 0.8;
prices["cykoria"] = 0.51;
prices["ogórek"] = 0.35;
prices["por"] = 0.10;
prices["śliwka"] = 0.06;
prices["czosnek"] = 0.01;
prices["ananas"] = 3;
prices["kaupsta"] = 0.30;
```

Figure 16 price map fragment

Loop iterate over every elements. When given element matches to element in *prices* map, value from the right side is added to *total_cost*.

```
strs << total_cost;                           // double to string conversion
std::string total_cost_in_string = strs.str();
```

Figure 17 Double to string conversion

The value produced from computing total cost is double type. Conversion to string is required. *Say()* method can read only string values.

[20] *C++ Documentation* Dec. 2015, URL: http://www.cplusplus.com/

```
OrderDetails od;

auto street_name = Recognize("Nazwa_ulicy", street_grammar);
od.street_name = street_name.text;

auto flat_number = Recognize("Numer_mieszkania", amount_grammar);
od.flat_number = street_name.text;

auto house_number = Recognize("Nazwa_bloku" , amount_grammar);
od.house_number = street_name.text;
```

Figure 18 Delivery details

To store data about delivery another structure *OrderDetails* was made.

```
struct OrderDetails {
    std::string street_name;
    std::string flat_number;
    std::string house_number;

};
```

Figure 19 OrderDetails structure

Information provided by client are inserted to particular fields to later use.

Either structures and map are inside *IVRDialogLogic.h* file.

```
Say("Kupiles", Beep::No);       //transaction summary
for (auto& a : Order){
Say(a.product, Beep::No);
Say("Sztuk",Beep::No);
Say(a.amount, Beep::No);
}
Say("Za kwotę",Beep::No);
Say(total_cost_in_string, Beep::No);

Say("Wysylka ulica", Beep::No);
Say(od.street_name,Beep::No);
Say("Na numer_mieszkania", Beep::No);
Say(od.flat_number,Beep::No);
Say("Blok numer", Beep::No);
Say(od.house_number,Beep::No);
Say("Do widzenia", Beep::No);
}
```

Figure 20 Transaction summary section

Last part is responsible for inform client about all order details and nicely say goodbye.

3. Conclusions

Like it was stated at the beginning speech processing has applications in almost every sphere of human life. Many decades of researches result in nowadays intelligent and complicated speech recognition systems. Something what forty or fifty years ago was treated as high Artificial Intelligence today is the norm. This technology is the resultant of many science branches like programming , electronics, signal processing and acoustic.

Speech processing engines are complex and necessary systems for all of the adhibition of speech processing. It is the fundamental on which many famous applications like Siri or Cortana[21] works. The creation process of such system is complicated and could take a long time.

Modern technology offers various and very easy ways to make telecommunication traffic between peoples or machines. Internet replaces telephone communications gradually in the manner of easier access to others. The big advantage of communication over internet like Voip is significantly lower call costs. What is more Voip is flexible, good for many applications

Voice interfaces are very common and comfortable way to create interaction between human and machine. Depends on application this kind of system can help to solve many kinds of problems. Reduction of outgoes , handling many clients at the same time and handling on distance are the ones of many pros.

Thanks to nowadays solutions process of designing voice interfaces is not complicated. Having the base of ASR, write proper code under concrete purpose it is not big challenge. To create basic voice interface even high programming skills are not required.

Software for speech sample generation are very helpful in voice interface design. Recording them by using microphone is quite difficult. To obtain reliable and not noisy records good equipment , special room and good

[21] *Speech Recognition Through the Decades: How We Ended Up With Siri* Dec. 2015, URL: http://www.pcworld.com/article/243060/speech_recognition_through_the_decades_how_we_ended_up_with_siri.html

speaker are required. A lot of time, precision, effort and money have to be invested to obtain such goal.

It is observable that project described in this document is not complex. A lot of corrections can be added. First of all extends the vocabulary to cover wider field of applications. The next thing is to create better dialog logic which would be more intuitive and more human friendly. For this improvements consist two exemplary things:

1. Write more verifications to make client sure what he is going to buy.
2. Create more methods which can improve functionality.

Bibliography

[1] *Speech Recognition Through the Decades: How We Ended Up With Siri* Dec. 2015, URL: http://www.pcworld.com/article/243060/speech_recognition_through_the_decades_how_we_ended_up_with_siri.html

[2] *Speech recognition* Dec. 2015, URL: https://en.wikipedia.org/wiki/Speech_recognition

[3] *Deep learning* Dec. 2015, URL: https://en.wikipedia.org/wiki/Deep_learning

[4] Juang B.H, Rabiner Lawrenece R. *Automatic Speech Recognition - A Brief History of the Technology Development* , 2004, Dec. 2015, URL: http://www.idi.ntnu.no/~gamback/teaching/TDT4275/literature/juang_rabiner04.pdf

[5] *Voip access* Dec. 2015, URL: https://www.happycall.pl/

[6] *Speech Recognition Grammar Specification Version 1.0* Dec. 2015, ULR: http://www.w3.org/TR/speech-grammar/

[7] *Augmented Backus - Naur Form* Dec. 2015, URL: https://en.wikipedia.org/wiki/Augmented_Backus%E2%80%93Naur_Form

[8] *Voice over IP* Dec. 2015 ,URL: https://en.wikipedia.org/wiki/Voice_over_IP

[9] *Session Initiation Protocol* Dec. 2015, URL: https://en.wikipedia.org/wiki/Session_Initiation_Protocol

[10] *C++ Documentation* Dec. 2015, URL: http://www.cplusplus.com/
[11] *Boost library* Dec. 2015, URL: http://www.boost.org/

[12] Christoper Schmandt , *Voice Communication with Computers: Conversational Systems* , 1993, Van Nostrand Reinhold , 0442239351

[13] Mihelic France , Zibert Janez ,*Speech Recognition* , 2008, InTech , 9789537619299

[14] Stroustrup Bjarne *The C++ Programming Language , 4th Edition* , 2013 , Addison-Wesley, 9780321563842

[15] *Cepstral Analysis of Speech* Dec. 2015, URL: http://iitg.vlab.co.in/?sub=59&brch=164&sim=615&cnt=1

[16] *Hidden Markov Model* Dec. 2015, URL: https://en.wikipedia.org/wiki/Hidden_Markov_model

List of figures