University “Politehnica” of Bucharest

Faculty of Electronics, Telecommunications and Information Technology

Automation System based on Microcontrollers

Diploma Thesis

submitted in partial fulfillment of the requirements for the ***Degree of Engineer*** in the domain ***Electronics and Telecommunications***, study program ***Technology and Telecommunication Systems***

Thesis Advisor   Student

Prof. Ph. D. Corneliu BURILEANU   **Elena Diana ȘANDRU**

2014

University "Politehnica" of Bucharest
Faculty of Electronics, Telecommunications and Information Technology
Department TELECOMMUNICATIONS

Department Director's Approval :

Prof. Silviu CIOCHINĂ, Ph. D.

[illegible]

DIPLOMA THESIS
of student ŞANDRU Gh. D. Elena-Diana, 441G

1. Thesis title: **Automation System based on Microcontrollers**

2. The student's original contribution will consist of (not including the documentation part):
The purpose of this thesis is the design and implementation of an automated system based on microcontrollers. For the practical realisation, the microcontrollers characteristics and the peripherals involved (such as sensors and engine) will be taken into account and also linked to the design and implementation of the software program that enables controlling the system by voice commands.

3. The project is based on knowledge mainly from the following 3-4 courses:
Microcontrollers, Microprocessor Architecture, Digital Signal Processing, Computers Programming

4. The construction part of the project remains in the property of : UPB

5. The Intellectual Property upon the project belongs to: Student, Thesis Advisor, UPB

6. The research is performed at the following location: Computer Room 3, UPB

7. The thesis project was issued at the date: 1.10.2013

THESIS ADVISOR: **STUDENT:**

Prof. Corneliu BURILEANU, Ph. D. Elena-Diana SANDRU

[illegible] [illegible]

Statement of Academic Honesty

I hereby declare that the thesis **"Automation System based on Microcontrollers"**, submitted to the Faculty of Electronics, Telecommunications and Information Technology in partial fulfillment of the requirements for the degree of Engineer/Master of Science in the domain **Electronics and Telecommunications**, study program **Technology and Telecomunications Systems**, is written by myself and was never before submitted to any other faculty or higher learning institution in Romania or any other country.

I declare that all information sources sources I used, including the ones I found on the Internet, are properly cited in the thesis as bibliographical references. Text fragments cited "as is" or translated from other languages are written between quotes and are referenced to the source. Reformulation using different words of a certain text is also properly referenced. I understand plagiarism constitutes an offence punishable by law.

I declare that all the results I present as coming from simulations and measurements I performed, together with the procedures used to obtain them, are real and indeed come from the respective simulations and measurements. I understand that data faking is an offence punishable according to the University regulations.

Bucharest, July 2014

Elena Diana ȘANDRU

Șandru

(student's signature)

TABLE of CONTENTS

LIST of FIGURES

LIST of TABLES

LIST of ACRONYMS

A

ABS-PC = Acrylonitrile Butadiene Styrene Polycarbonate

API = Application Programming Interface

ASR = Automatic Speech Recognition

C

CPU = Central Processing Unit

D

DCM = Device Control Manager

DFT = Discrete Fourier Transform

DOF = Degree Of Freedom

F

FSR = Finite State Grammar

FSR = Force Sensitive Resistors

G

GMM = Gaussian Mixture Models

H

HMM = Hidden Markov Model

I

IFT = Inverse Fourier Transform

IMAP = Internet Message Access Protocol

L

LAN = Local Area Network

LM = Language Model

LPC = Linear Predictive Coefficients

M

MFCC = Mel-Frequency Cepstrum Coefficients

MP = Mega Pixels

MRE = Magnetic Rotary Encoder

N

NSR = Network-based Speech Recognition

O

OOV = Out Of Vocabulary

P

PA-66 = PolyAmide 66

PC = Personal Computer

PCM = Pulse Code Modulation

PDF = Probability Density Function

PLP = Perceptual Linear Prediction

PNG = Portable Network Graphics

POP3 = Post Office Protocol version 3

S

SER = Sentence Error Rate

SMTP = Simple Mail Transfer Protocol

SOC = System-On-Chip

SSL = Secure Sockets Layer

T

TCP = Transmision Control Protocol

W

WAN = Wide Area Network

WAV = Wave Audio File Format

WEP = Wired Equivalent Privacy

WER = Word Error Rate

WPA = Wi-Fi Protected Access

X

XML = eXtensible Markup Language

INTRODUCTION

Our world is constantly changing and evolving and the technological developments are increasing spectacularly. Phrases such as "Century of Speed" or "Information Century" are well known nowadays; the reality is that we live in an era in which, at first, man became dependent on machines due to their ability to facilitate work. Currently, the population growth, the need for communication, globalization, scientific developments have led this man dependency on technology to another level. The challenge of having a society life, rest periods or the help needed with daily chores, pushed technology as indispensable in daily life.

Soon, the lack of time has become a major problem, together with the human convenience, so they tried to simplify interaction with machines, to make it faster and easier. So did the idea of voice control and automatic speech recognition (ASR) appeared; nowadays, the speech recognition feature is available on more and more devices. Smartphones, tablets, wearable devices respond to the voice commands people spoke. This feature is gaining popularity and specialists predict that in the future every interaction with a machine will be based on vocal commands, due to its natural characteristics.

At the same time, man is regarded as a social being, a being who likes to be surrounded by other people, or in the last decade of intelligent machines that behave humanly and resembles the human body. So, the development and evaluation of interactive humanoid robots that communicate with humans and are designed to participate in human society as partners was the next step of technological evolution.

Field of speech recognition has been extensively developed by the international scientific community, such as performing ASR systems have been implemented for many languages. The particularities of the Romanian language made very difficult the development of an efficient ASR; considerable effort to acquire the necessary resources for Romanian language was made by the Speech and Dialogue Research Laboratory team. Thus, an ASR was designed and implemented for our native language.

NAO is a biped autonomous humanoid robot, developed by Aldebaran Robotics. It was first released in 2006 and with each generation, it evolved to become everybody's friend, as Aldebaran-Robotics says about it [3]. Aldebaran created NAO to be a true daily companion; its humanoid shape, its behavior and its face features recommends it as a great tool for smart houses and even more. NAO was built as a research humanoid robot, but the latest versions were improved in order to make it suitable for homes; the company tagline is "One NAO in every home".

Thus, combining two important technological trends nowadays, namely the Automatic Speech Recognition for Romanian language and NAO, the humanoid robot, is the strongest motivation for choosing the theme of this thesis.

Following the observations made above, the main objective of this thesis is designing and implementing the architectural solution and the necessary applications for NAO to respond to voice commands spoken in Romanian language. In order to do this, a number of specific objectives were considered:

- ✓ Understanding of the construction principles of the existing ASR, as well as the communication protocol between any client and the server on which ASR is implemented, and the creation of models (acoustic, phonetic, linguistic) for the proposed solution
- ✓ Setting the optimum principle of the robot programming.
- ✓ Developing the auxiliary application that links the robot and the server.

This thesis is structured in four chapters; first two shows the theoretical aspects of an ASR and the engineering and programming particularities of NAO. The last two chapters reveal the author's contribution to the development of the practical part of this thesis. The personal contribution can be summarized as follows:

- ✓ Creating an achievable architectural solution for the main objective;
- ✓ Establishing the commands list for NAO, given the particularities and the performances of the ASR;
- ✓ Deciding the proper solution to be implemented on the robot – a timeline behavior;
- ✓ Designing and creating the Choreographe behavior (AutomationSystemSpeechRecognition.crg), application running on NAO and making it perform the following actions: record the vocal command, send it through e-mail (as an attachment) to a certain recipient, wait while the speech-to-text transcription is conducted, fetch the e-mail for the command sent as plain text, perform the indicated action;

- ✓ Deciding the use of a reliable programming language for the application development;
- ✓ Designing and creating a Java application (MainApplicatin.java) with the following capabilities: access the e-mail server (corresponding to the recipient account), fetch the e-mail message that was previously send by the robot and save the attachment (representing in fact the vocal command), send data to the server (using the existing protocol), receive the transcription of the vocal command (transmitted data) and send via e-mail plain text message the speech-to-text sequence back to the robot;
- ✓ Adapting the existing communication protocol to the server and the existing application to the MainApplication.java.

CHAPTER 1

NAO – Autonomous Humanoid Robot

A humanoid robot is in fact a robot having the shape similar to the human body. Various reasons can be declared regarding this feature, but the most important ones will be highlighted: the humanoid design has functional purposes (interact with human tools) and experimental purposes (study the bipedal locomotion and other features of the human body) [1].

Recent research concentrated also on androids, humanoid robots that resemble perfectly the human body and also act like a human. These realistic robots are no longer related to the science fiction domain, as they were two decades ago.

The term “autonomous” refers to the ability of the robot to perform several tasks with a certain degree of autonomy; autonomy means independence of control. Furthermore, it represents a property of the relation between the designer and the autonomous robot. Self-sufficiency, learning or development, and evolution increase an agent’s degree of autonomy [2].

NAO is a biped autonomous humanoid robot; its body consists of two arms, one head, a torso and of course, two legs. NAO’s construction along with the NAOqi operating system make NAO a versatile robot, used in several domain such as education or medicine.

1.1 NAO’s General Characteristics

NAO is a 58-cm tall humanoid robot. It was first released in 2006 and with each generation, it became better and better, evolving to become everybody’s friend, as Aldebaran-Robotics says about it [3]. Aldebaran created NAO to be a true daily companion; its humanoid shape, its behavior and its face features recommends it as a great tool for smart houses and even more.

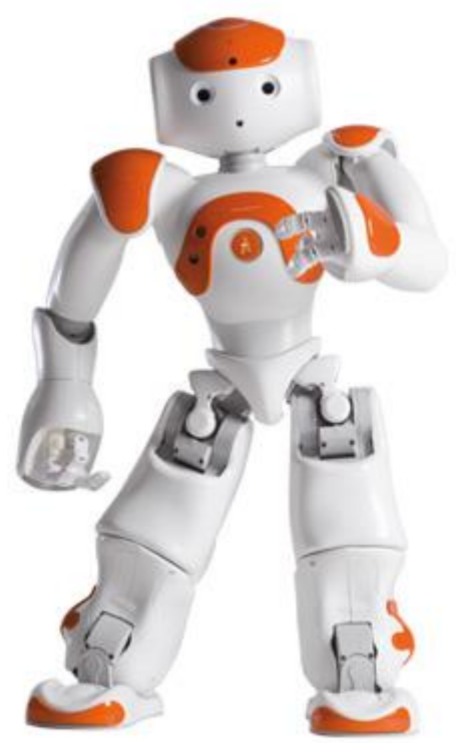


Figure 1.1 Aldebaran – Robotics NAO humanoid robot [3]

As figure 1.1 shows, NAO resembles very accurate the human body. This innovative robot is lightweight, compact and can be also cataloged as a mobile robot. Two important characteristics that distinguish NAO from other existing humanoids are the very mobile joints and the pelvis kinematics design.

Another important feature is represented by the domains in which it can be used. NAO is currently used in the education field in over 70 countries, in 300 universities; the research domains that use NAO are also expanding nowadays. The next stage for NAO is to be included in every home, as a help for the parents, as a friend for the kids, as an interface with the new concept of smart homes. This robot was designed to be affordable without sacrificing quality and performance [4].

1.2 Hardware Features and Mechatronic Design

NAO can be characterized by the following words: affordability, modularity and performance, from hardware point of view. NAO's hardware platform has been built from the ground by the company engineers.

A. CONSTRUCTION and ELECTRICAL Features

NAO's dimensions are: 574x275x311mm, while its weight is around 5.2 kg. The construction material is a combination between ABS-PC and PA-66; the Polycarbonate-ABS is widely used in industry and has the advantage of being flexible (ABS) but without losing strength (PC), while Polyamide 66 is a versatile thermoplastic [5]. Dimensions and weight influence directly the motion performance.

$$\text{BMI} = \frac{weight\ [kg]}{height[m]^2} = \frac{5.2}{1.14} = 4.56 \qquad (1.1)$$

As it can be seen in equation 1.1, NAO's BMI is around 4.56 which makes it very light compared with other competitors (so it is less susceptible to breakdown). NAO is equipped with a Lithium-Ion battery, providing a 48.6Wh; front the autonomy point of view, it can be further improved. The data regarding this feature are the following: 60 minutes – active use and 90 minutes – normal use.

B. SENSORS

All humanoid robots are equipped with sensors; these devices help it to communicate to the environment, measuring some attributes of the surrounding areas. The sensors fall into two categories:

- ✓ Proprioceptive – orientation, speed and position of the robot.
- ✓ Exteroceptive – tactile, sound and vision sensors.

Sensor type	No. of sensors on NAO
Camera	2
Speaker	4
Microphone	2
FSR	8
MRE	36
Gyrometer	1
Accelerometer	1
Infrared sensor	2
Ultrasonic sensor	4
Contact sensor	9

Table 1.1 List of NAO's sensors

Table 1.1 lists the sensors that are integrated in NAO. In the following pages, all of them will be detailed.

- ✓ Vision

NAO is equipped with 2 cameras, characterized by MT9M114, System-On-Chip (SOC) image sensor; the resolution is about 1.22MP. The cameras have integrated both functions: recording and taking picture. The picture format in PNG, lossless image compression format, while for video recording the frames per seconds depend on resolution and in the case of video stream, strongly dependent on the speed of the communication standard (Gigabit Ethernet, 100Mb Ethernet, Wi-Fi). The main disadvantage of the vision feature is represented by the placement of the 2 cameras (on the forehead). This leads to the vision impossibility in the case of very close objects.

- ✓ Audio

The audio feature includes 2 speakers and 4 microphones, all placed on the head. The frequency range for microphones is 150Hz-12kHz, while for speakers is up to 20kHz.

- ✓ Sonar

The robot has 2 ultrasound devices (emitter and receiver) situated in chest. They provide space information, the detection range being between 0.05 meters up to 3 meters, if an object is situated at 30 degrees from the robot chest (60 degrees all cone combining both devices). The assembly works at a frequency of 40 kHz.

- ✓ IR

Device IR is divided in two parts, the infrared (IR) emitter and the infrared sensor. It emits and detects at a wavelength equal to 940 nm, at an angle of +/- 60 degrees.

- ✓ Inertial Unit

NAO has a gyrometer and an accelerometer. These sensors are two important devices when we are talking about motion, helping us to find if the robot is in a stable position or in unstable one when is walking. The accelerometer measures the proper acceleration, while the gyrometer (gyroscope) measures orientation, based on the angular momentum principles. Both devices have 3 axes and a precision from 1 to 5 %.

- ✓ FSR

Force Sensitive Resistors sensors are made from a material whose resistance is changing when applying a force, most of the time a conductive polymer. FSR are used to create pressure-sensing areas, NAO having 4 of them on each foot. The force range of a FSR is 0-110 N.

- ✓ Position

The position sensors integrated in NAO are called Magnetic Rotary Encoder (MRE). The principle is the following: the magnetic sensor reads the position of the encoder, established by the position of the two or more magnetic poles.

✓ Contact

The contact sensors are reunited into: Chest Button, Foot Bumper, Tactile Head and Tactile Hand. All these sensors let us know if the robot is touching something, in different parts of its body (such as foot, head or hand).

C. ACTUATORS

The actuators are the motors in charge with the movement of the robot. For the humanoid robots, the actuators act like muscles and joints. NAO's motors are DC motors and figure 1.2 shows their location. The legend is the following one: Joint Name [Motor Type] [Reductor Type]. Everything will be detailed in table 1.2.

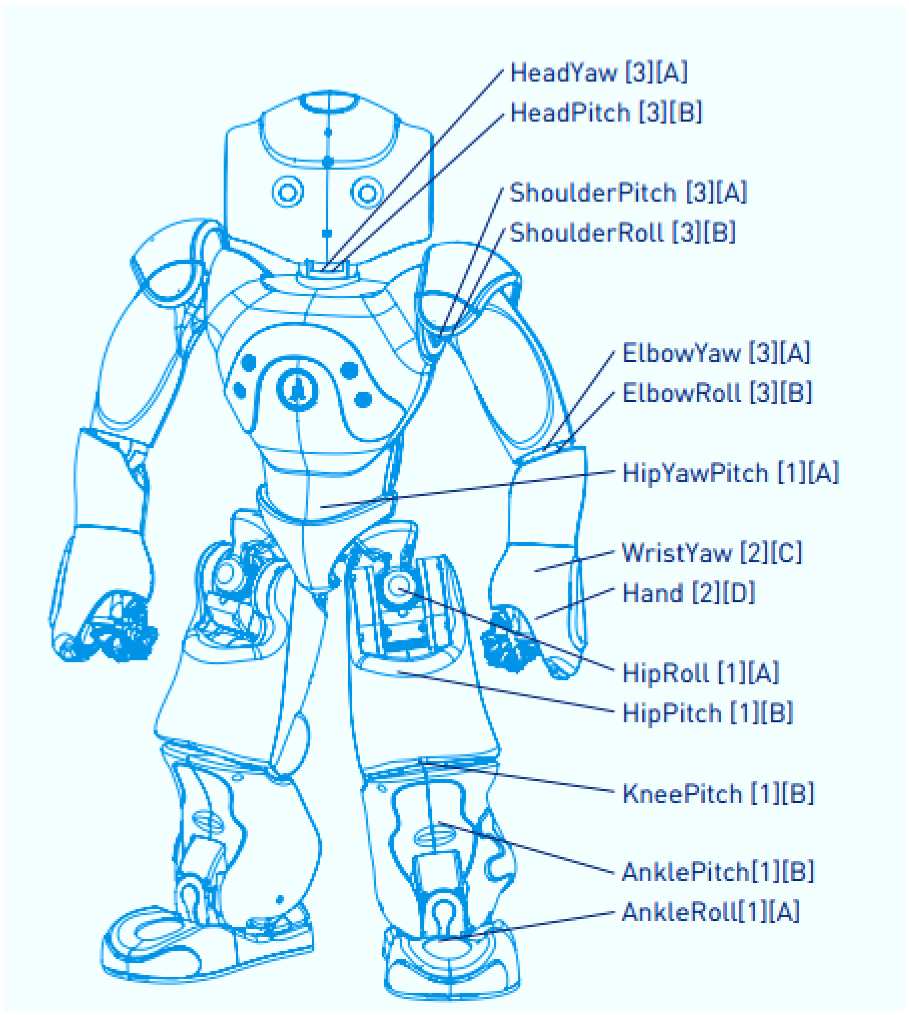


Figure 1.2 NAO's motors [5]

✓	✓ Joint Name	✓ Motor
Head Joints	HeadYaw	Type 3
	HeadPitch	Type 3
Arm Joints	ShoulderPitch	Type 3
	ShoulderRoll	Type 3
	ElbowYaw	Type 3
	ElbowRoll	Type 3
	WristYaw	Type 2
	Hand	Type 2
Leg Joints	HipYawPitch	Type 1
	HipRoll	Type 1
	HipPitch	Type 1
	KneePitch	Type 1
	AnklePitch	Type 1
	AnkleRoll	Type 1

Table 1.2 Position of Motors [5]

NAO's motor type can fall into 3 categories:

- ✓ Type 1 – speed: 8300rpm ±10% and stall torque: 68mNm±8%
- ✓ Type 2 – speed: 8400rpm ±12% and stall torque: 9.4mNm±8%
- ✓ Type 3 – speed: 10700rpm ±12% and stall torque: 14.3mNm±8%

It can be observed that Type 1 represents a powerful motor, able to sustain the entire body of the robot, that's why this type is used for the legs; furthermore, its speed is not very big, the main reason being the fact that it must be able to provide stability, not speed. The head, the shoulders and the elbows are used very much in most of the actions and the motors used for these areas must confer a higher speed and a better torque compared to the ones used for wrists and hands.

D. MOTHERBOARD

NAO has two CPUs: one located in the head, which runs a Linux kernel and another one located in the torso.

The first CPU, the one located in the head is equipped with an Intel ATOM Z530 processor, 45nm Core dimension, with 512KB cache memory, based on x86 architecture. The clock speed is 1.6GHz, and the Front Side Bus (the one connecting the CPU to the Northbridge) is 533MHz. The instruction set is 32-bit and the microcontroller supports the Intel System Controller Hub (Intel® SCH), being a single-chip component designed for low-power operation, but still a single-core processor for mobile devices offering enhanced performance. [6]

ATOM Z530 is a powerful microcontroller and the ratio between performances and price is very good. Its main advantages can be the fact that is power efficient and it also supports hardware acceleration of virtualized applications, being an ultra-low-voltage CPU. This microcontroller is based on the Bonnell microarchitecture, being able to execute up to two instructions per cycle. It translates the CISC instructions into simpler internal operations (RISC instructions) prior to execution.

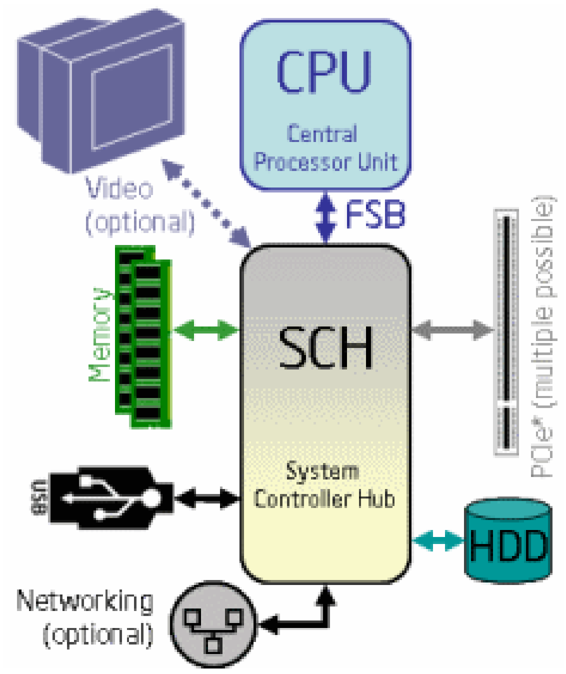


Figure 1.3 ATOM Z530 diagram [6]

The second CPU is an ARM7TDMI microcontroller that controls the actuators, by distributing information to all actuators module microcontrollers. ARM7TDMI is a 32-bit embedded processor, widely used in nowadays designs. One of its advantages is the fact that supports 16-bit instructions via the ARM and Thumb instruction sets.

The actuators module microcontrollers are Microchip 16-bit dsPICS and the communication between second CPU and them is performed through two RS485 buses: one that connect ARM7TDMI and the upper part and another one for the lower part of the body (throughput of 460Kbps).

The ARM7TDMI microcontroller communicates with the CPU board through a USB-2 bus with a theoretical throughput of 11Mbps.

E. DOF

The Degree Of Freedom (DOF) represents the number of independent parameters that define the robot design (state). NAO has 25 DOFs, divided as table 1.2 shows: 11 for the lower part (legs and pelvis) and 14 for the upper part (head, arms and hands):

Location	No. of DOFs
Head	2
Arm	5 (in each)
Leg	5 (in each)
Hand	1 (in each)
Pelvis	1

Table 1.3 NAO's DOFs [5]

Each leg has 2 DOFs at the ankle, 1 DOFs at the knee and 2 DOFs at the hip, and each arm features 2 DOFs at the shoulder, 2 DOFs at the elbow, 1 DOFs at the wrist and 1 additional DOF for the hand's grasping. The head can rotate about yaw and pitch axes [4].

F. CONNECTIVITY

NAO currently supports Wi-Fi and Ethernet, the most widespread network communication protocols. It is compatible with the IEEE 802.11b/g/n Wi-Fi standard and can be used on WPA (Wi-Fi Protected Access) and WEP (Wired Equivalent Privacy) networks, requiring no Wi-Fi setup other than entering the password. Regarding Ethernet, the robot is fitted with a RJ-45 plug, accepting 3 adaptations: 10/100/1000 BASE T, (speeds of 10Mbps, 1000Mbps or 1Gbps and twisted pair cable - the pair of wires for each signal is twisted together to reduce radio frequency interference and crosstalk between pairs).

Figure 1.4 sums up the entire hardware (from electronics point of view) structure of the robot, including the links between sensors/actuators and microcontrollers. The main components are the microcontrollers:

- ✓ CPU Board located inside the head
- ✓ ARM Microcontroller placed in the chest
- ✓ dsPICs Microcontrollers present next to the actuators.

The figure displays the area of influence of each component from the above. From example, the ARM microcontroller is in charge with commanding the dsPICs, which command further the motors. Linked to the CPU are the components regarding vision, audio, IR, connectivity and LEDS.

dsPIC microcontrollers are related to the motors, MRE sensors, LEDs and to the infrared feature.

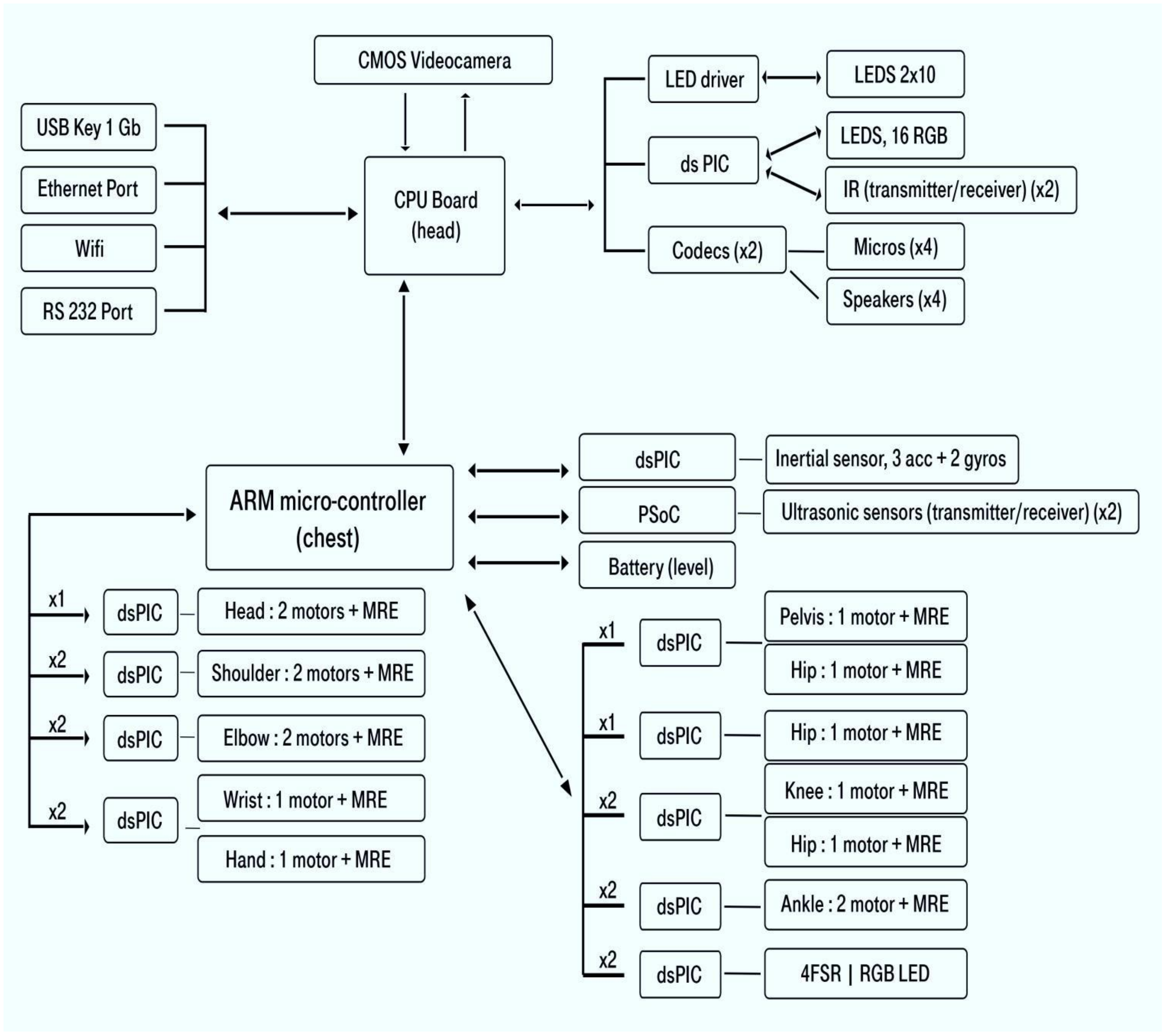


Figure 1.4 NAO’s electronics architecture

1.3 Software Characteristics

NAO's main software is called NAOqi; it runs on the robot and controls it. The main characteristic of the software is the fact that it is distributed, meaning each component can be executed locally (robot's on-board system) or distributed between systems, while NAOqi Daemon is running on the main system. The limitations come when using actuators and sensors; the main system - Main Broker, must be executed on NAO [3].

The programming framework is called NAOqi Framework, allowing homogenous communication between different modules [7]. The notion module will be defined in the following paragraphs. NAOqi framework has three properties [3]:

✓ Cross-platform – one can develop with it on Windows, MAC or Linux.
✓ Cross-language – software can be developed in C++ or Python; the existing API can be called from any language.
✓ Introspection – capabilities, monitoring and action on monitored functions performed by API; it knows all the functions available. Unload a library and the robot won't be able to perform the functions linked to that library.

NAOqi can be divided in three parts [7] as the figure 1.5 shows:

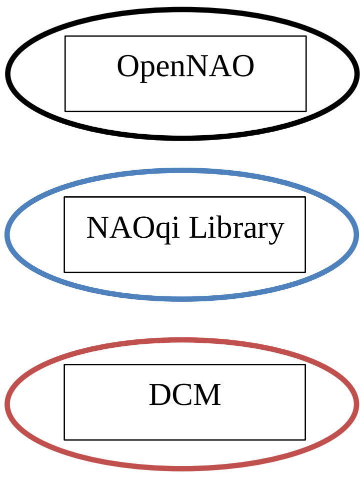


Figure 1.5 NAOqi Platform

1. NAOqi Operating System or OpenNAO – an embedded GNU/Linux (on-board) distribution developed to be executed on NAO. The operating system provides programs and libraries, being the software that gives life to the robot.
2. NAOqi Library – is divided into objects; those objects contain some actions that NAO is able to perform. The main advantage of this library is the abstraction layer; its functionality allows the user to program the robot without accessing directly the hardware. Figure 1.6 shows the objects found in this library. It can be seen that DCM is also a part of the mainBroker.

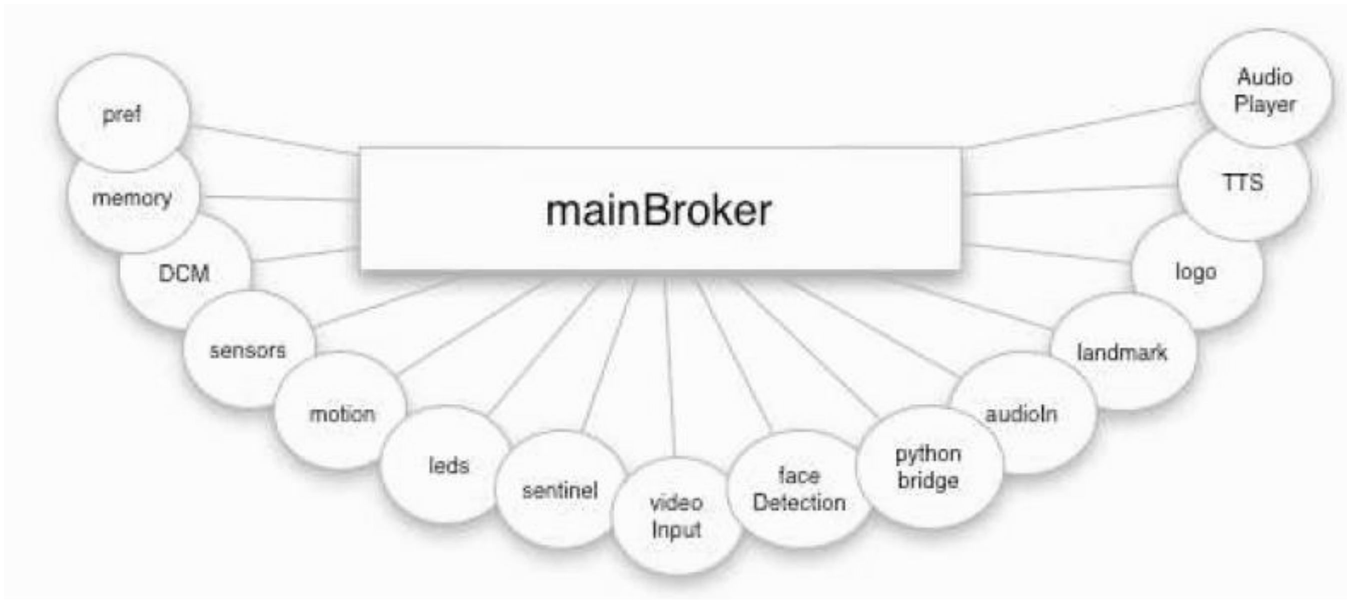


Figure 1.6 NAOqi library [7]

3. Device Control Manager or DCM – represents a set of libraries. The difference between it and NAOqi Library consists in the processor that receives the function calls; DCM sends directly to the ARM microcontroller of the robot.

NaoQi is an Object Oriented Programming, which minimum object element is called Module (usually a class within a library). NaoQi is an event-based programming and the modules interact one to another using shared memory (ALMemory) [7]. The NAOqi executable which runs on the robot is a Broker, having two roles: providing directory services (allowing the user to find modules and methods) and providing network access (allowing the methods of attached modules to be called from outside the process).

When the broker starts, it loads a preferences file called autoload.ini, defining which libraries should be load. Each library contains one or more modules that use the broker to advertise their methods. Loading modules forms a tree of methods attached to modules, and modules attached to a broker, as figure 1.7 displays. For example, methods attached to AlMemory library are: insertData(), getData(), RaiseEvent(), etc.

Another instance is the Proxy; it is defined as an object that will act as same as the module it represents. In order to have access to all the methods of a module, the user must create a proxy to that module.

A more intuitive approach to program NAO is using the GUI software developed by the company, called Choregraphe, which comes with a series of behavior macro-blocks. Programming NAO using Choregraphe is discussed in Chapter 3, where the behavior running on the robot is developed and implemented.

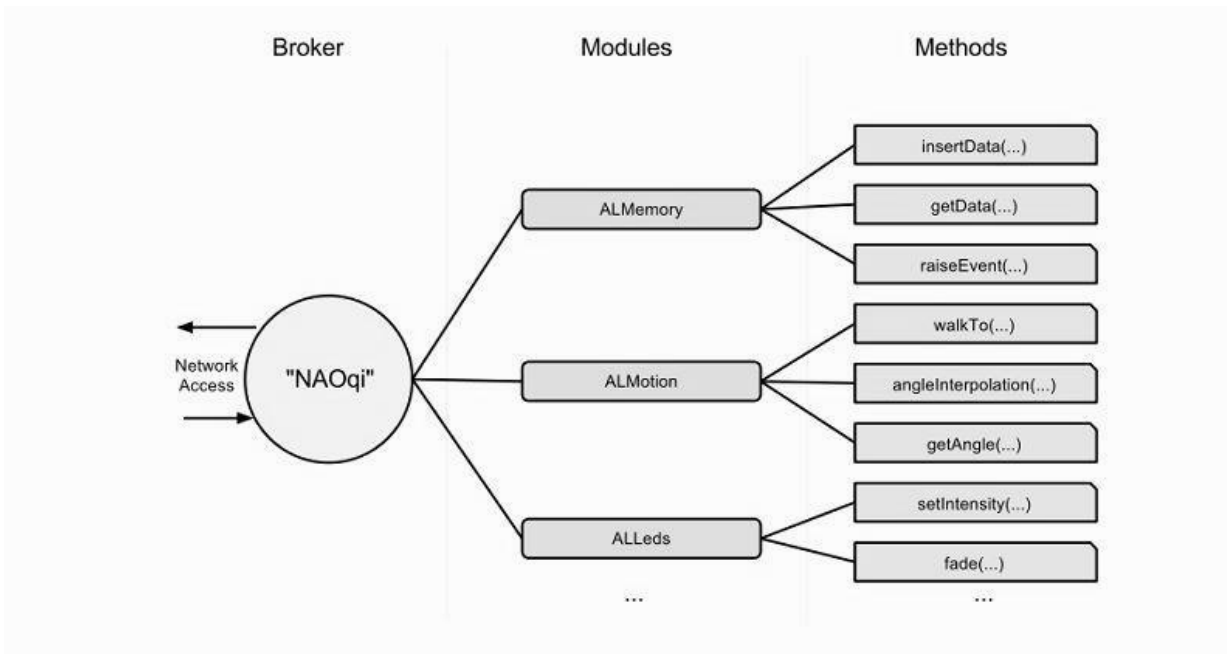


Figure 1.7 Broker-Modules-Methods [3]

CHAPTER 2

Automatic Speech Recognition

The process of Automatic Speech Recognition (ASR) can be defined as the independent, computer-driven transcription of spoken language into readable text in real time [9]. In the last 50 years, the speech research concentrated on developing a machine able to understand fluently spoken speech and although the nowadays technology is far from understanding the entire speech, in any environment and spoken by any person, impressive progresses were made.

The final goal of ASR research is to allow the machine to recognize in real time, independently on the noise, vocabulary dimension, speaker characteristics and accent, with one hundred percent accuracy, all the words spoken by a person. In 1975, Baker put the basic of statistical framework for ASR – the DRAGON system, a system that makes systematic use of a general abstract model to represent each of the knowledge sources necessary for automatic recognition of continuous speech [10] , followed by a team from IBM in 1976 and another one from AT&T in 1983.

2.1 Fundamentals of Speech Signal Formation

Spoken language is used to communicate information from a speaker to a listener [11]. Studies showed that not only the speech production is important, but also the speech perception, a key element in the speech chain.

2.1.1 Mechanism of Speech Production

Speech is the acoustic end product of voluntary, formalized motions of the respiratory and masticatory apparatus [12]. It represents the main form of communication between human beings and as it will be presented in this paper, even between human beings and machines. The speech signal, and moreover the vocal message are composed of elementary units called phones (analog sound patterns) representing the basis for spoken language representation, such as syllables and words.

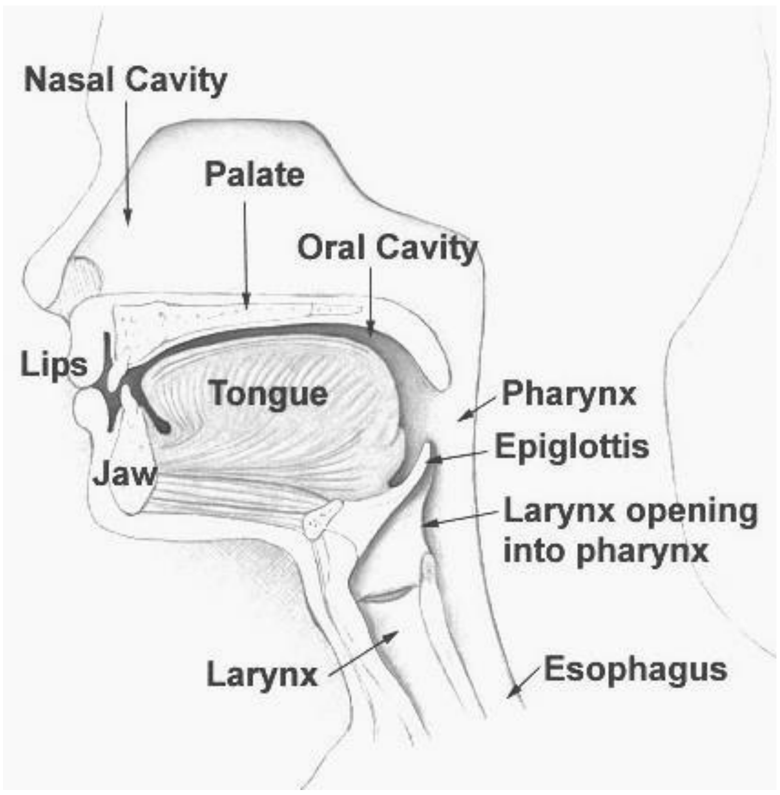


Figure 2.1 Schematic of speech apparatus [13]

From the content point of view, a vocal message can be characterized by a strict or a wide sense. The strict sense gives the message intelligibility, ideally being the same as the written message, while the wide sense takes into consideration also the message intonation, offering in this way a more powerful meaning to the vocal message than the written message.

The human speech apparatus is situated between the lungs and the lips. The respiratory apparatus provides the necessary energy for speech production, under the form of waves of air pressure, when the airflow is pressed out of the lungs, then pushed out through the trachea. After that, it is modulated by the larynx before entering the vocal tract.

The larynx represents an ensemble of muscles and mobile cartilages which surrounds a cavity situated at the upper part of the trachea. Placed across the larynx are situated the vocal cords (2 pairs), the entire system manipulate the pitch and the volume (intensity) of the generated sound. The capability of the vocal folds to oscillate together or not, defines the speech sound as being voiced or un-voiced. The glottis represents the place where the vocal folds come together and its dimension is controlled by their contraction. In this way, the glottis modulates the air flux that passes through it.

The vocal tract is composed of the oral tract (mouth and pharynx) and the nasal tract (within the nose). Here, the air will undergo further changes as it makes its way upwards towards the mouth. Different types of sounds can be generated by only controlling parts of the vocal tract, which represents in fact a succession of acoustic tubes and resonant cavities.

2.1.2 Phonetic Characteristics

The vocal message can be decomposed in small and distinct sounds named phones. The phone is the basic unit of phonetic speech analysis and represents the smallest unit in speech signal by which two words can be different (physical or perceptual). For example, the difference in meaning between the Romanian words “cad” and “car” is the result of the exchange between the phones [d] and [r].

The main problem in learning a new language or creating an acoustic model for it appears due to the fact that one cannot establish a rigorous letter-phone correspondence. In the same time, a phone can be represented by several letters while a letter can correspond to more than one phone.

A standardized classification of phonemes in Romanian language (as in any other language) does not exist; in what follows will be presented a possible classification. Phones can be divided into three categories, depending on the way they were produced by the vocal tract:

- Consonant – the speech sound formed with the partial or complete enclosure of the human vocal tract. It can be divided into the following subclasses:
    - ✓ Nasal consonant – for their pronunciation both oral and nasal cavities are used (at the same time).
    - ✓ Fricative consonant – can be split into:
        - o Unvoiced
        - o Voiced – implies the vibration of the vocal cords.

    - ✓ Obstructive consonant – for their pronunciation, the air is obstructed upstream of the larynx during the "retention time" and then it is suddenly released. Can also be split into:
        - Unvoiced
        - Voiced - implies the vibration of the vocal cords.
    - ✓ Liquid consonant – combines lateral and vibrant sounds.
- Semivowel – the speech sound similar to a vowel (phonetically point of view) but characterized as being non-syllabic, meaning it is rather the syllable boundary than the syllable nucleus.
- Vowel – the speech sound articulated with an open human vocal tract. It is characterized as being syllabic.
    - ✓ Oral – the nasal tract is not involved in the pronunciation.
    - ✓ Nasal - for their pronunciation both oral and nasal cavities are used (at the same time)

2.2 ASR Architecture

The task of a speech recognition system is to return a sequence of words when at its input is applied an acoustic signal. The systems based on Hidden Markov Model see this task as a noisy channel and the acoustic signal is a version of the words sequence affected by the noise. This affects the recognition performances.

The problem can be formulated as follows: determine the way the noisy channel affects the source and after that determine the source sequence by finding the best match between the original sequence and the sequences memorized in the system, passed through the noisy channel. Therefore, the problem of speech to text conversion can be summarized as: "Given the input acoustic sequence X, what is the most likely sequence of words $\widehat{W}$ in the language L?"

The sequence X represents a sequence of individual observations, or otherwise said, consecutive samples, where each index $\mathrm{x_i}$ is a time interval:

$$\mathrm{X = x_1, x_2, \ldots, x_t} \qquad (2.1)$$

The sequence W represents a sequence of words and $\mathrm{w_i}$ is an individual word:

$$\mathrm{W = w_1, w_2, \ldots, w_n} \quad (2.2)$$

In this moment the task can be quantified using the *arg max* function, which selects the argument that maximizes the probability of the word sequence. So, the most probable sequence ($\widehat{W}$) is the one with the highest a posteriori probability (P (W|X)) (knowing the input acoustic sequence X).

$$\widehat{W} = \arg max_W \mathrm{P(W|X)} \qquad (2.3)$$

Using Bayes criterion for computing the a posteriori probability, the equation becomes:

$$\widehat{W} = \arg max_W \frac{\mathrm{P(X|W)P(W)}}{\mathrm{P(X)}} \qquad (2.4)$$

In the equation 2.4, the term P(X) is the probability of the input acoustic sequence, independent of the word sequence W, therefore it can be neglected. The new equation will be:

$$\widehat{W} = \arg max_W \ \mathrm{P(X|W)P(W)} \qquad (2.5)$$

The problem of speech to text conversion is even simpler now: instead of estimating the word sequence given the input acoustic sequence, we should estimate two quantities: the a priori probability of the word sequence W (P(W)) and the likelihood of the acoustic information given the word sequence (P(X|W)).

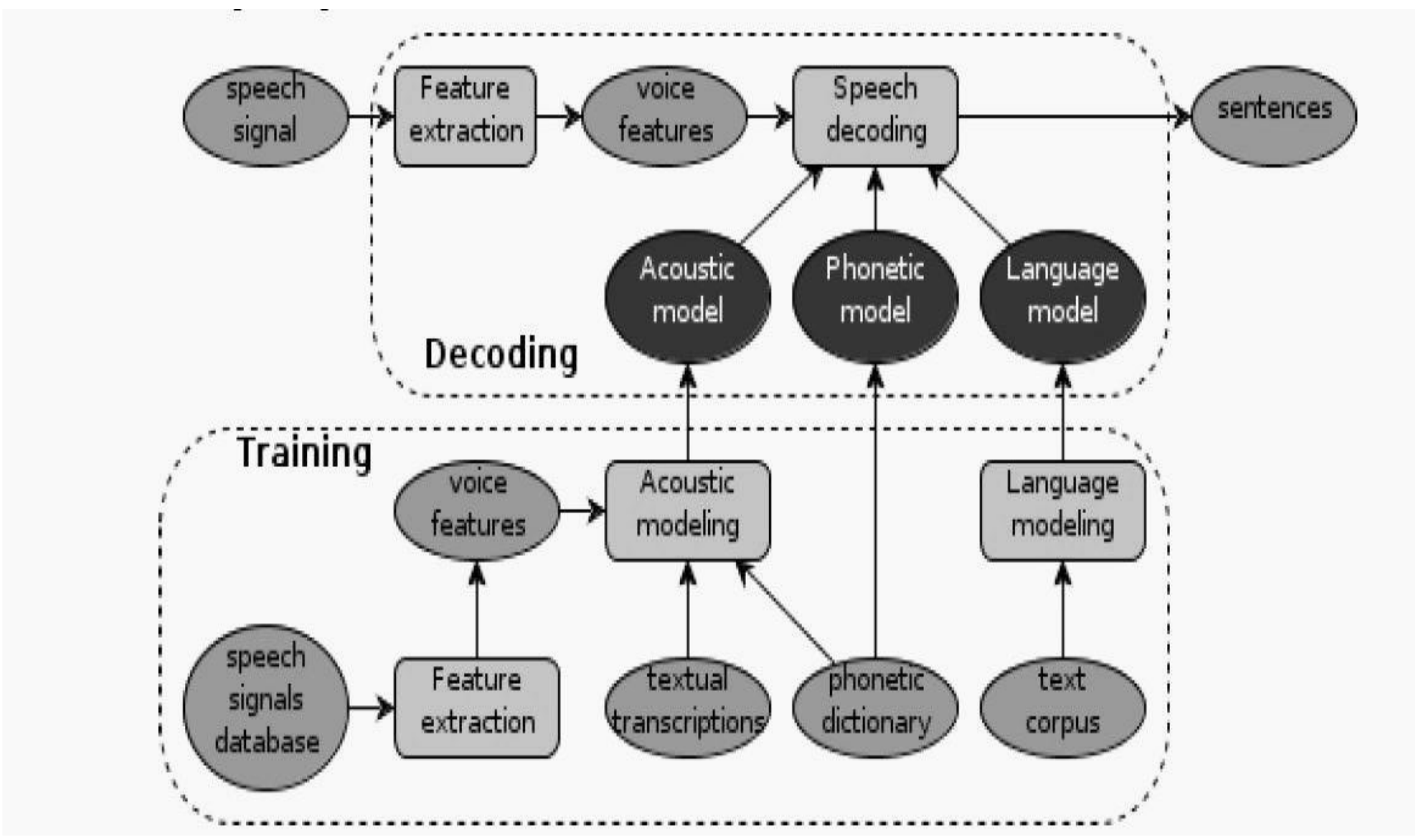


Figure 2.2 ASR System Architecture [8]

The figure 2.2 exhibits the architecture of ASR system. Based on the language model, the probabilities of the words in a given set can be estimated. The language model should introduce constraints due to the fact that in any language some phone sequences appear more often than others. The acoustic model helps estimate the probability of the acoustic sequence, given a word sequence. In any language, the number of distinct words can be approximated as large and this is why in the acoustic model the basic unit is not the word, but the phone.

As the figure 2.2 shows, these two models can be built separately, but they work together to decode an input acoustic sequence as stated in the equation 2.5. Besides this, a third model must be implemented in order to connect the language and the acoustic models, meaning the phonetic model. In fact, the phonetic model is a phonetic dictionary, associating to each word in the existing vocabulary its phonetic representation.

2.2.1 Acoustic Modeling

The Acoustic Model (AM) can be described as the nucleus of the ASR and not only once the term recognition was related directly with this model. It uses acoustic models trained before and the input acoustic sequence parameters in order to decide the basic unit language (phone) which was pronounced by the speaker.

In this manner, AM is composed of a set of phones models and in the decoding stage of the process are linked to form words in the first place and word sequences after. Acoustic modeling is based on the Hidden Markov Models (HMM), which provides an effective way to integrate, in an unified manner segmentation, time mapping, matching shapes and context.

2.2.1.1 Acoustic features extraction

As figure 2.2 shows, HMM does not use the input acoustic speech waveform (time-domain) in order to perform the recognition, but a series of features extracted from the original sequence. The Feature Extraction and Voice Extraction block perform these operations.

The digital signal represents a signal composed of a sequence of discrete values. A digital vocal sound is in fact a representation of the changes in pressure during the speech.

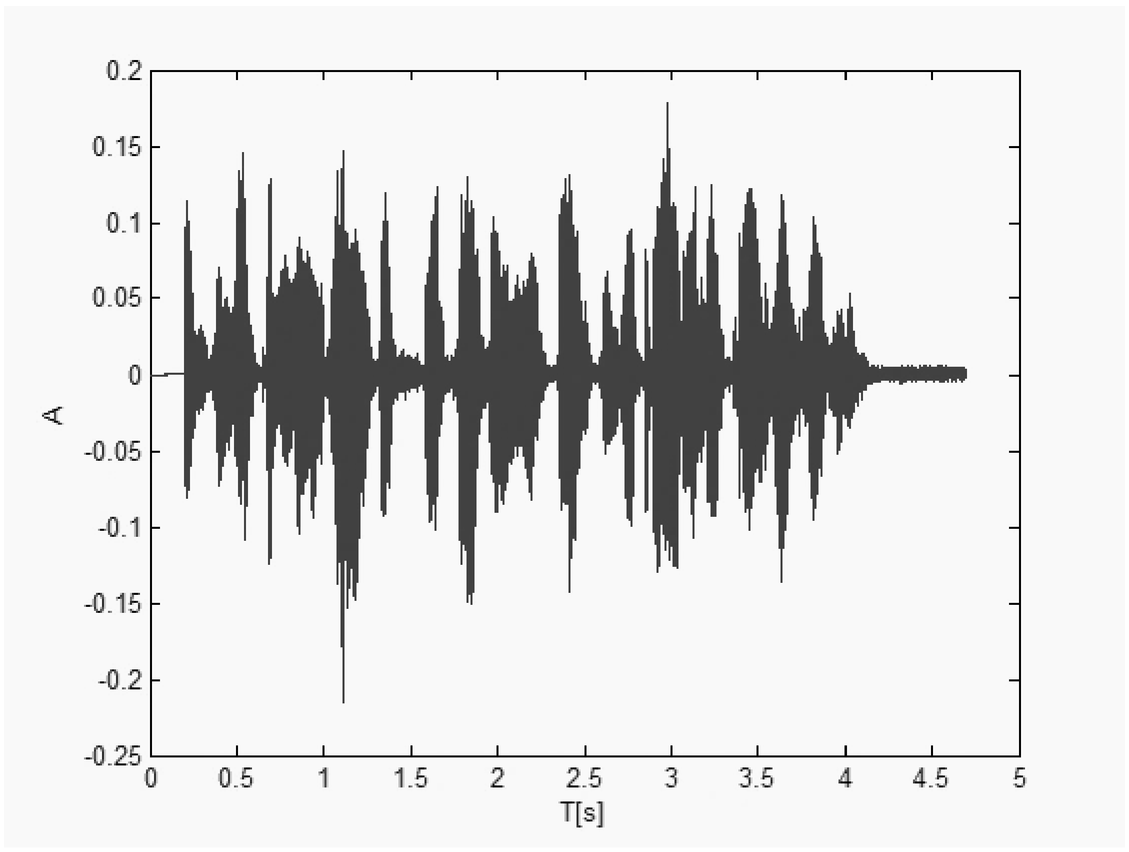


Figure 2.3 Uttered sentence “Campionatul mondial de fotbal 2014 are loc în Brazilia”

Figure 2.3 shows the time-domain waveform of a vocal sound [the code can be found in Appendix 1]. The time-domain waveform is composed from several consecutive samples, strictly related to the sampling frequency (for this particular signal the sampling frequency is F_S = 44.1kHz) and to the signal length.

A time domain representation shows the variations of the signal over time. In speech recognition domain (and not only), the frequency representation is used because it offers more information (regarding phase and how much of the signal lies within a frequency) and is easier to work with.

The most common frequency-domain representation is the one based on Discrete Fourier Transform (DFT). A way to plot the DFT is by using the spectrogram, a 2-dimension representation in time-frequency domain.

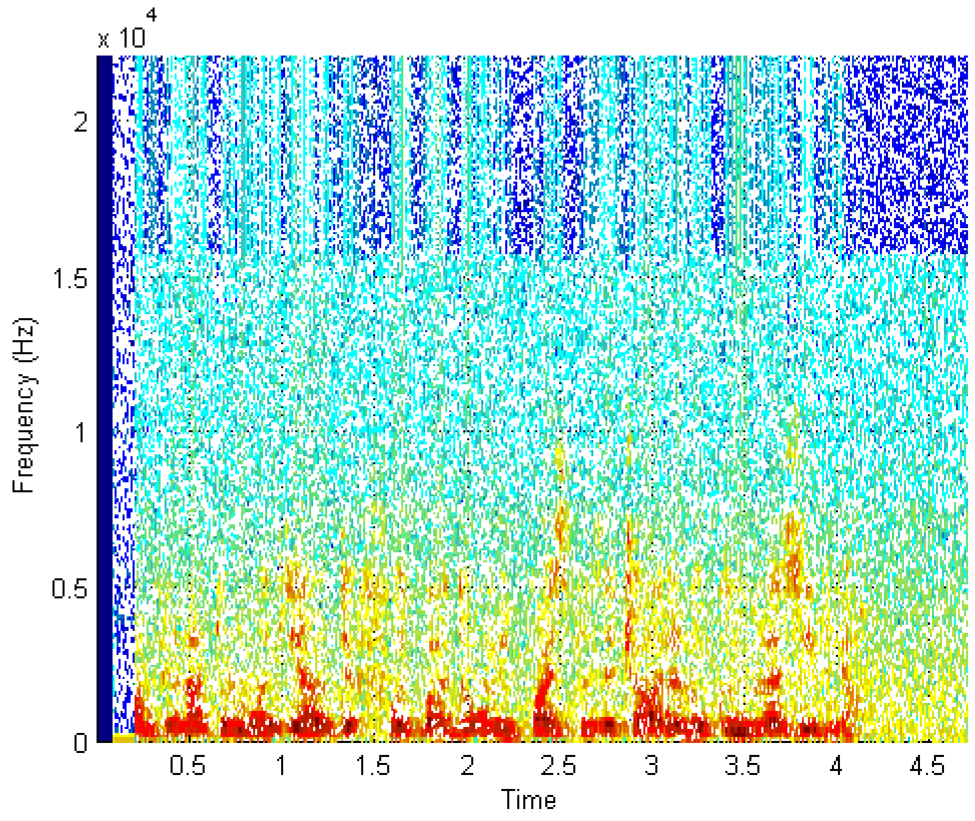


Figure 2.4 Spectrogram of the signal presented in figure 2.3

The point (x,y) represents the power of the signal at frequency y and the time moment x. The red color indicates the formants. The audio signal and thus the vocal signal are un-stationary signal, but they have the great property to be stationary during small amounts of time. This characteristic represents the main reason for the vocal sound to be divided in frames of 20 to 30 ms. The assumptions that a 30 ms frame is stationary is stated. These frames are generated at about 10 ms, leading to a 10-15 ms overlap between consecutive frames.

The border between consecutive frames would introduce important artifacts in spectral domain so a Hamming window is applied to the signal before DFT. To sum up, the time-domain signal is subjected to a framing followed by a windowing process and the result is a stationary signal.

The voice parameters used in the recognition domain can be:

- ✓ Linear Predictive Coefficients (LPC) – based of least-squared methods assumed on all-pole model (autoregressive model) [14]
- ✓ Mel-Frequency Cepstrum Coefficients (MFCCs)
- ✓ Perceptual Linear Prediction (PLP) - uses three concepts the critical-band spectral resolution, the equal-loudness curve and the intensity-loudness power law. The auditory spectrum is then approximated by an autoregressive all-pole model [15].

MFCCs were introduced by [16] and have the big advantage of being uncorrelated, compared to the spectral coefficients, highly correlated one to another (neighboring coefficients).

The human auditory system perceives the sounds in a logarithmic manner and the acoustic features are not linearly distributed on the frequency scale. Thus, MFCCs are based on a linear cosine transform of the logarithmic power spectrum on a nonlinear Mel frequency scale. Firstly, the cepstrum is computed by finding the IFT of the logarithm of the estimated spectrum of the speech signal. Also, cepstrum is a homomorphic transform, the convolution of two signals in time-domain is the sum of their cepstra in the cepstrum-domain.

The main difference between cepstrum coefficients and Mel-frequency cepstrum coefficients is closely related to the spacing of the frequency bands: linear spacing for normal cepstrum and equally spacing on mel scale for MFCC, as the following equation shows:

$$f_{\mathrm{Mel}} = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \qquad (2.11)$$

The figure 2.5 exhibits the way MFCCs are computed. The procedure is quite simple: the speech signal is passed through framing and Hamming windowing (as stated above), then DFT is applied in order to obtain spectrum. The Mel scale is divided into equal frequency bands and to each frequency band corresponds a triangular band pass filter. The powers of the spectrum are now mapped on Mel-scale and the logarithm of the power at each Mel-frequency is computed. The last step is to apply the DCT (Discrete Cosine Transform) at the resulting list of log powers. The MFC coefficients represent the amplitudes of the final spectrum.

Usually, only 12 to 20 coefficients are extracted after the computations. Commonly, ASR use a 39-dimension feature vector composed of 12 MFCCs plus the energy, and their first and also second derivative.

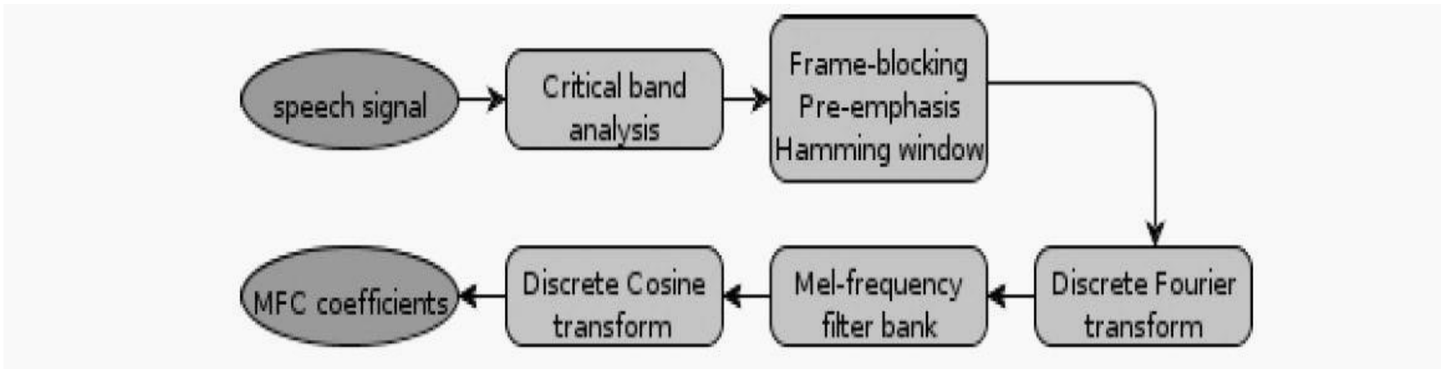

speech signal
Critical band analysis
Frame-blocking Pre-emphasis Hamming window
Discrete Fourier transform
Mel-frequency filter bank
Discrete Cosine transform
MFC coefficients

Figure 2.5 Block diagram for MFCC's computation [8]

2.2.1.2 HMM framework

As was stated at the beginning of this sub-chapter, the acoustic modeling is based of HMM. We discuss further about the HMM framework used in ASR. The processes that take place in an ASR starting with the application of the input acoustic signal and ending with the return of the word sequences are presented in the next figure:

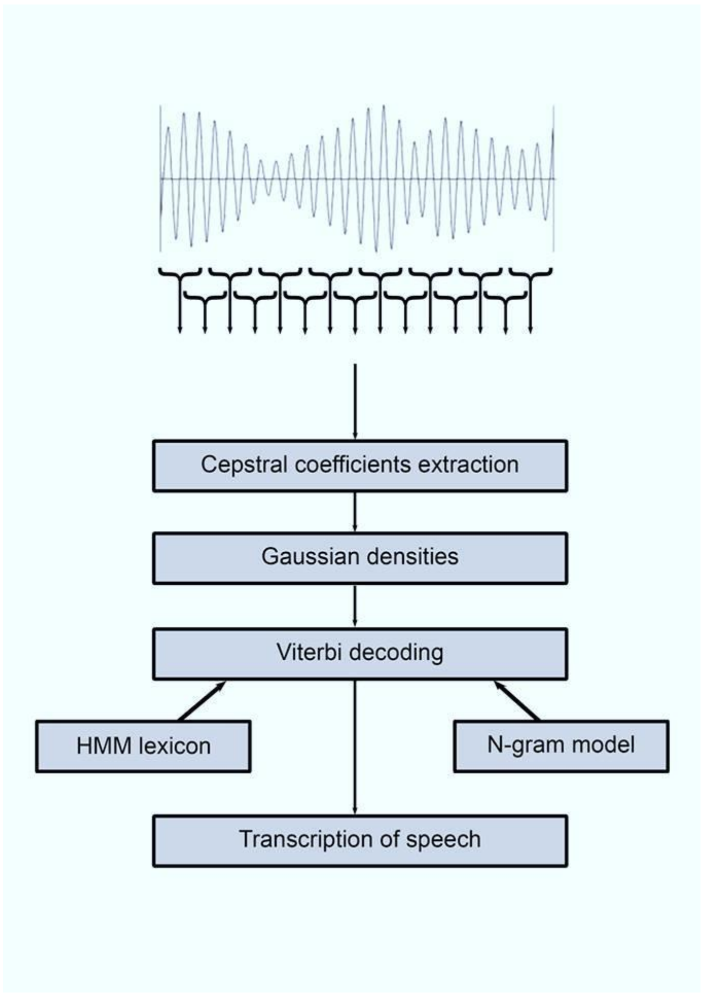


Figure 2.6 Simplified architecture of the decoding process in an ASR

We previously discussed about the MFCCs (cepstral coefficients extractions) so now we will focus on the Gaussian densities or otherwise named Gaussian Mixture Models (GMMs).

GMM is a parametric probability density function represented as a weighted sum of Gaussian component densities [17]. GMM parameters are estimated based on the training data and are highly capable to represent a large class of sample distributions. GMM uses a discrete set of Gaussian functions and each is defined by its covariance and mean matrix.

A HMM is a probabilistic finite state automaton, consisting of states connected by transitions. The term "hidden" derives from the fact that the states are not directly observable. Instead of the state sequence, an acoustic feature vector sequence is observed. A HMM is built in a hierarchic process, that starts from the words, decomposed into phones. To each phone a HMM is associated.

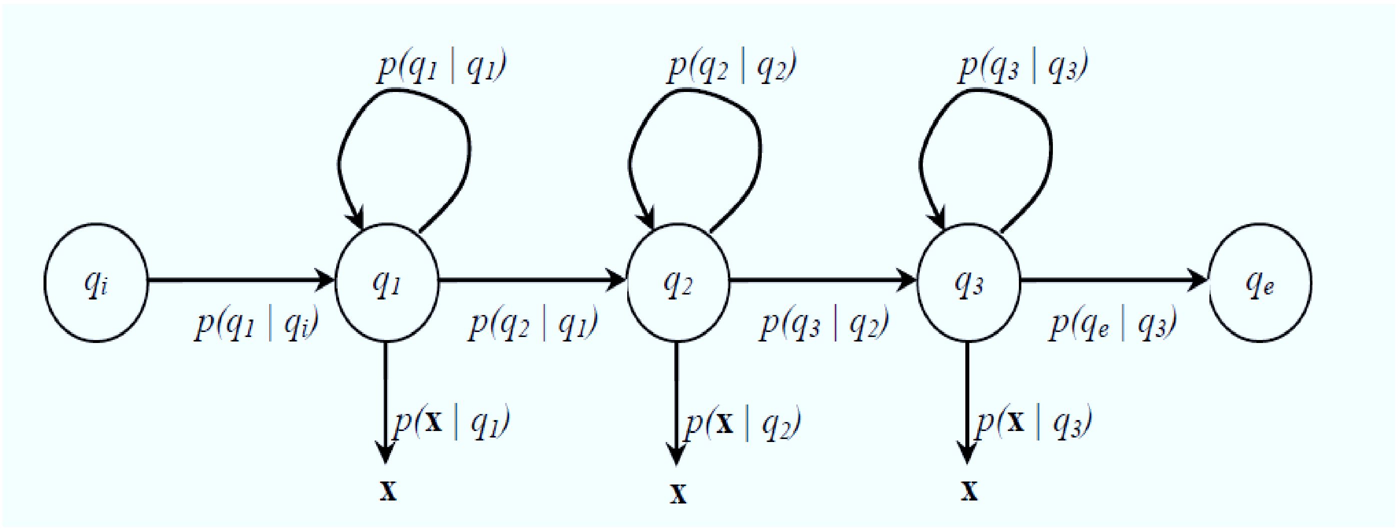


Figure 2.7 HMM – parameterized stochastic finite state automaton

Figure 2.7 shows that HMM has the following parameters:

- ✓ A set of States : $Q = q_1, q_2, \dots, q_N$
- ✓ A set of Transition Probabilities : $A = a_{11}, a_{12}, \dots, a_{NN}$, where $a_{ij} = p(q_j|q_i)$ is the probability of transitioning from state *i* to state *j*.
- ✓ A set of Observation Probabilities : $B = b_i(x_t) = p(x_t|q_i)$ which represents the probability of an observation x_t to be generated by the state *i*.

In speech recognition is not allowed the transition from one state to any other state, although the definition of the HMM states that. It is important to maintain the natural flow of the speech which is sequential so important constraints are attached to backward transitions. The auto-loop is not restricted, allowing a sub-phonetic unit to repeat in order to cover a variable amount of the acoustic input [8]. The actual model used in speech recognition system is the one presented in figure 2.7; it allows only auto-loops and/or transition to a successive state.

Besides the above parameters, an additional set can be introduced – the initial state probabilities π_i, defined as the probability of one state to be the first from a state sequence. Parameters π_i and A can be estimated during the training process.

The output of the HMM are acoustic features and can take a wide range of real values. If there is a finite number of possible observations, then the probability of observing state q_i can be considered a discrete function (although it is in fact a Gaussian PDF).

There are three main problems in working with HMM:

- ✓ Evaluation
- ✓ Decoding
- ✓ Estimation

The evaluation problem implies computing the probability to generate a sequence of observations, given the model and the sequence of observations. The modern ASR systems are based on HMM, taking into account two assumptions: firstly, the observations are independent and secondly, the state sequence is considered to be a first-order Markov process; these assumptions significantly diminish the computational complexity. Thus, the total probability to generate a sequence of observations equals the sum of all the states that might lead to the given sequence $X = (x_1, x_2, \dots, x_T)$. To easily compute this, the Forward Algorithm is used:

$$\alpha_t(q_j) = \mathrm{p}\,(x_1, x_2, \dots, x_t, q_t = q_j \mid \lambda) \qquad (2.12)$$

In equation 2.12, λ represents a wider set of parameters, composed of *A*, π_i and *PDF*. $\alpha_t(q_j)$ is the probability of observing the sequence of observations *X* and being in state q_j at moment *t*. Because we assumed the observations to be independent, the recursive formula for this probability is the equation 2.13.

$$\alpha_t(q_j) = \sum_{i=1}^{N} \alpha_{t-1}(q_i)\alpha_{ij}\mathrm{b}_j(x_t) \qquad (2.13)$$

The decoding problem can be formulated: find the most likely state sequence to have generated a sequence of observations, given the model and the sequence of observations. The Viterbi Decoding is applied; the Forward Algorithm would be very complex from the computational point of view – a similar structure algorithm is considered.

The Viterbi Algorithm returns the most probable sequence of states and the summation is not done at every step (as it was performed in the Forwarding Algorithm), but instead, a max operation is performed in order to remember the best path.

The estimation problem is the most complicated of the three issues stated above. We have no a priori knowledge about which state generated each sequence of observations. The simplest and efficient criterion is the maximum likelihood estimation in which the parameters are established in such way to maximize the probability of the observations sequence to be generated by the model.

The used algorithm is the Baum-Welch algorithm; each step is composed by two parts:

- ✓ Step E - a state-time alignment is performed, by assigning an occupancy probability of each state for each time moment (given the sequence of observations).

- ✓ Step M – the parameters are estimated by a weighted average of the occupancy probabilities of the states.

To sum up what was discussed in the 2.2.3 sub-chapter, ASR systems use HMMs with GMMs as output PDFs to model speech units (phones most of the time). The modeling is performed based on a series of perceptual acoustic features (MFCCs) derived out of the original time-domain input acoustic speech signal. The HMM parameters are estimated using Baum-Welch algorithm. The Viterbi algorithm is used to decode the sequence in order to find the most probable sequence of states given the sequence of observations. Usually, each speech unit has one HMM; the global HMM is formed by concatenating all basic HMMs.

2.2.2 Phonetic Modeling

The Phonetic Model (PM) also introduces restrictions in the decision process, knowing that not all phones combinations are possible. It was previously stated that the word is not the basic unit in an ASR. There can be highlighted two reasons for this measure: firstly, each recognition task can have new words, for which there isn’t sufficient training resource and secondly, the number of words in any language in very large.

Thus, it is important to perform a correspondence between each word and its phonetic representation. Most of the times, the PM is a pronunciation dictionary that performs the mapping word-phones sequence. This phonetic dictionary is a link between the Acoustic Model and the Language Model.

2.2.3 Language Modeling

The Language Model (LM) is a very important part of the speech recognition system because it introduces supplementary restrictions starting from the premise that not any word sequence is possible. These restrictions can include sets of grammar rules or statistical information regarding the word sequence, reducing in this way the number of possible combinations of word sequences.

LM is used during decoding process of the input acoustic sequence, having the role to estimate the probabilities of all word sequences from the searching space. In fact, the goal is to estimate as accurate as possible if a word sequence can be a sentence in that specific language. Furthermore, this can be summarized as making decisions regarding the order of the words and attaching probabilities such that natural word choices to be easier recognized.

One approach can be computing P(W) based on counting how many times the word sequence W occurred – disadvantageous from computational point of view. Another approach – the used one – is based on determine the probability of a word sequence by computing the probability of each word, based on a number of previous words, namely the N-gram Model. The probability of a sequence $W = w_1, w_2, \ldots, w_n$ can be expressed as:

$$P(W) = P(w_1, w_2, \ldots, w_n) = P(w_1)P(w_2|w_1)\ldots P(w_n|w_1, \ldots w_{n-1}) \qquad (2.6)$$

According to the equation 2.6, the task of estimating the probability of a word sequence W is divided into smaller task which estimate the probability of each word, knowing the previous word sequence. N, from N-gram Model indicates the number of previous words which affects the probability of current word. This number is chosen based on the amount of data used for training. The bigger the amount of training data is, the higher the order of the language model can be.

The frequent N-gram is the trigram, meaning the probability of the current word depends on two previous words. In order to build a N-gram model, one must count the number of occurrences of each N-word combinations in a sufficiently large text corpus. For example, in order to estimate the probability of occurrence of the sequence $w_i\, w_j\, w_k$ (using the maximum likelihood principle)

$$P(w_k|w_i, w_j) = \frac{count\,(w_i, w_j, w_k)}{\sum_w count\,(w_i, w_j, w\ \ )} \qquad (2.7)$$

The goal of a language model is to predict the next word based of previous known words, taking into account the redundancy of a language. Any N-gram must be evaluated in order to determine its performances and a series of metrics have been introduced:

✓ N-gram hits

An important issue that can appear in building and using N-grams is the data sparseness and an used method to undergo this disadvantage is the Back-Off method. The Back-off method supposes the usage of several language models in order to create an interpolated version of all (due to robustness of lower order N-grams). For example, if unigram, bigram an trigram are available, an interpolated language model can be built as a linear combination of the known language models.

The trigram hits measures the ratio between how many times the model could use the both previous words and how many times it had to back-off (to a bigram or even unigram) in order to find the probability of the n-gram:

$$\text{trigram hits [\%]} = \frac{\text{number of triagram hits}}{\text{number of words}} \times 100 \qquad (2.8)$$

✓ Perplexity

A language model can be evaluated regarding the value of the probability it assign to a word sequence. Ideally, it should assign a high probability to a good sequence and a low probability to a not so good sequence.

The perplexity is the reciprocal of the geometrical average probability assigned to each word from the word set W, by the language model:

$$PP(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|w_1, \ldots, w_{i-1})}} \qquad (2.9)$$

Generally, a lower value of perplexity is correlated to a better performance of the language model, leading to less confusion in speech recognition.

- ✓ Out of Vocabulary Words (OOV)

Although in most of the situations the vocabulary is closed (the test-set contains only words from the vocabulary), in real life there are many words which are not in the vocabulary and consequently they cannot be predict by the language model. The OOV rate represents the percentage of the words which are not in the vocabulary. Because the perplexity of these words is infinite, it cannot be summed up in the total perplexity and the best way to evaluate a language model is by specify the OOV rate along with the perplexity:

$$\text{OOV}[\%] = \frac{\text{number of OOV's}}{\text{number of words}} \text{ x } 100 \qquad (2.10)$$

2.3 ASR Evaluation

Evaluating an ASR system is in fact a comparison between the reference (the correct word) and the word provided by the system. The standard metric is the Word Error Rate; in order to compute WER one must compute the minimum distance between the resulted word and the reference.

$$\text{WER [\%]} = \frac{\text{No.of Insertions+No.of Substitutions+No.of Deletions}}{\text{No.of total words in correct transcript}} \text{x100} \qquad (2.14)$$

The equation 2.14 gives the formula of WER as the ratio between the sum of number of word substitutions, insertions and deletions (necessary to map between reference and word provided by the system).

There are cases when Sentence Error Rate (SER) is more useful:

$$\text{SER [\%]} = \frac{\text{No.of sentences with at least one erroneous word}}{\text{No.of sentences in correct transcript}} \text{x100} \qquad (2.15)$$

CHAPTER 3

Speech Recognition Application

Nowadays, the speech recognition feature is available on more and more devices. Smartphones, tablets, wearable devices respond to the voice commands people spoke in order to find what they want and need. This feature is gaining popularity and specialists predict that in the future every interaction with a machine will be based on vocal commands.

Unfortunately, the ASR needs a high calculus power that most of the time cannot be implemented on such device. NAO is gifted with an ASR for English language, but implementing one for Romanian language is a complicated task and requires features that NAO does not have (such as calculus power).

NAO was built as a research humanoid robot, but the latest versions were improved in order to make it suitable for homes; the company tagline is “One NAO in every home”. NAO will be a very pleasant interface between human and smart houses, connecting it to the operating system that controls the house feature being the next step.

Nevertheless, the first phase is programming NAO to perform a series of actions as a result of vocal commands spoken in Romanian language.

3.1 Architectural Solution

The main objective of this diploma thesis is for a user to command the humanoid robot NAO using vocal commands, pronounced in Romanian language. So, the purpose is developing a solution for the text-to-speech transcription inside or independent on the robot. Currently, it does not exist an ASR for Romanian language available for any user. Because of the language specifications, it is a very hard task to develop such system. Considerable effort to acquire the necessary resources for Romanian language was made by the Speech and Dialogue Research Laboratory team.

As it was discussed previously, an ASR requires a high calculus capacity for signal processing and all the functions that are implemented inside it. Due to this important drawback, the solution proposed for vocally command NAO is a NSR system. A Network-based Speech Recognition system involves the following:

- ✓ Acquisition of the vocal signal in performed at Client level
- ✓ Decoding of the vocal signal in performed at Server level

The next phase in finding the architectural solution is to establish the connection between the robot and the server. The server used in this project is described in the Chapter 4. Worth mentioning now about it is the fact that the communication protocol is based on sockets (and also XML messages). The choice was to maintain the same protocol with the server and design the client part (as part of the Client – Server Architecture).

The Client level was divided into two distinct parts:

- ✓ The Robot, as the input and the output of the entire system. At its level, the acquisition of the vocal signal is done and the purchased file is send to a Java application; after the speech-to-text transcription, the transcript of the vocal command is received by the Robot and the proper action is performed.
- ✓ The Java application, as the client part from the Client – Server application. It must establish the connection with the server, send the file, receive the transcription and send it to the Robot.

The architectural solution proposed by the author is presented in figure 3.1 and it involves two types of communication: one through LAN (between the Robot and the Laptop) and another one through WAN (between the Laptop and the Server).

The flow of the diagram is the following one: the user pronounce a command, the robot records it and then sends the resulted .wav file using the e-mail to a specified recipient. A Java application performs a pooling action, interrogating periodically the server that hosts the e-mail account the .wav file has been sent to and when a new e-mail is received (containing the acoustic signal as an attachment), that e-mail it fetched, the attachment saved and the connection with the server is established.

The audio data is sent to the server (as a bytes stream), the speech-to-text transcription is performed and the resulted text data is received by the application. The transcription of the vocal command is sent back to the robot using the e-mail (plain text). The respective account in now fetched by the robot, the transcription is extracted and NAO performs the action indicated by the command.

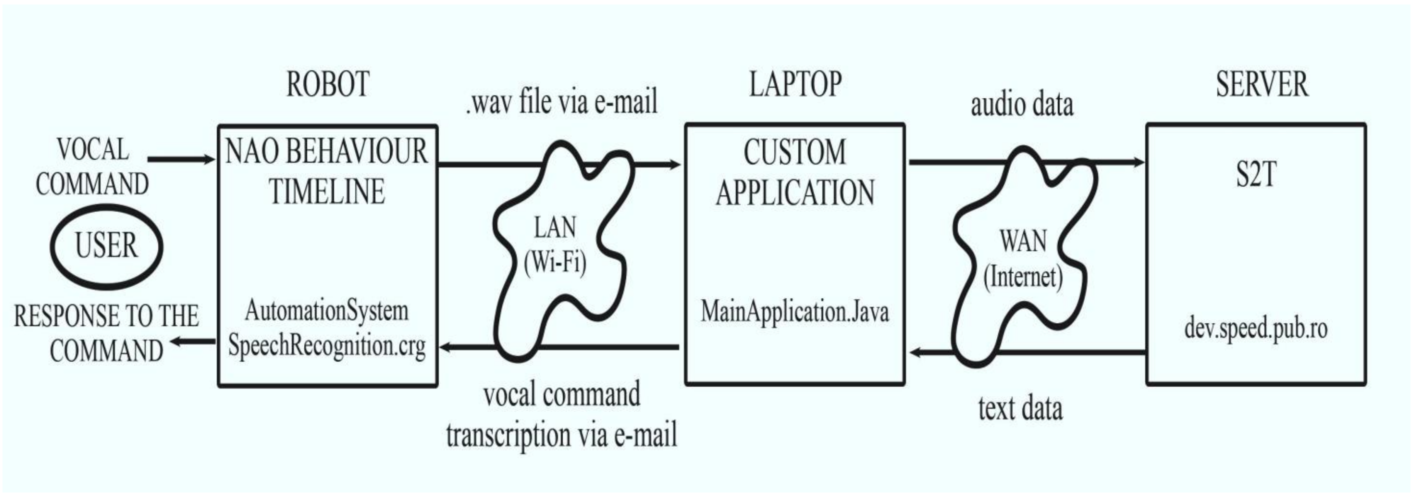


Figure 3.1 Architectural solution

The contribution of the author is concentrated on two plans:

- ✓ The Robot - NAO: Establish the list of orders, implement the behavior timeline for NAO in order to properly respond to the commands.
- ✓ The Custom Application: Interrogate the e-mail account for new e-mails, fetch the new e-mail and save the attachment on the computer, communicate with the server in order to send the audio file, get the response (the transcription) and sent it back to the robot as plain text e-mail.

The Custom Application (MainApplication.java) will be discussed in Chapter 4. During this chapter the discussion concentrates on the contribution regarding NAO.

The author of this thesis established a simple commands list composed of five commands that mostly imply simple movements of NAO. The list was designed in such way that the words to be as different as possible one from another in order for the ASR to not introduce errors while transcription is performed. The list is the following:

- ✓ “Stai jos”
- ✓ “Mergi un metru”
- ✓ “Mișcă mâna dreaptă”
- ✓ “Cine sunt eu?”
- ✓ “Ridică-te în picioare”

On the other hand, NAO must: record the vocal command, send it via e-mail, wait for the response, retrieve the e-mail received from the application and perform the previously command. The entire flow forms a behavior, the AutomationSystemSpeechRecognition.crg file that is loaded on the robot. This behavior is developed using Choregraphe software.

3.2 Choregraphe Programming

Choregraphe is a multi-platform desktop application that allows creating animations and behaviors, testing them on a simulated robot, or directly on a real one, monitor and control NAO [18]. Choregraphe allows creating very complex behaviors, without writing code lines, due to the macro-blocks that are already implemented; same time, these functions are open-source and can be modified as the user wants.

Choregraphe makes the interaction with NAOqi easier. The user can create quickly animations and programs for NAO that would be way longer to do with NAOqi alone. Choregraphe is an easy and intuitive GUI software. Predefined functions are stored in libraries, and are grouped based on the module they use, as figure 3.2 shows; the macro-blocks in Box Library (Standard or Advanced). The script of the macro-block is available to the user and the programming language is Python.

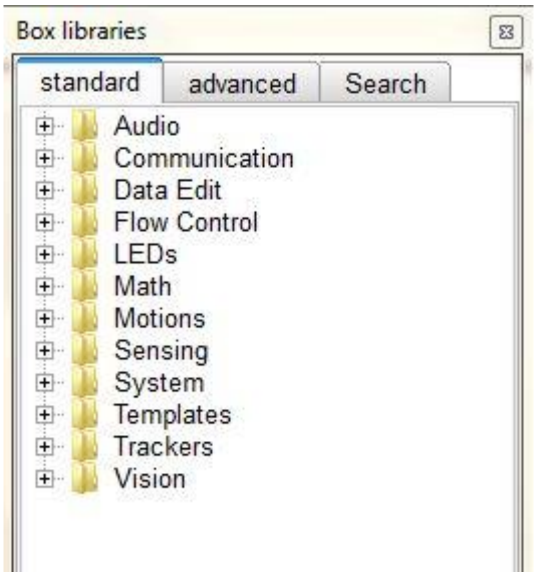
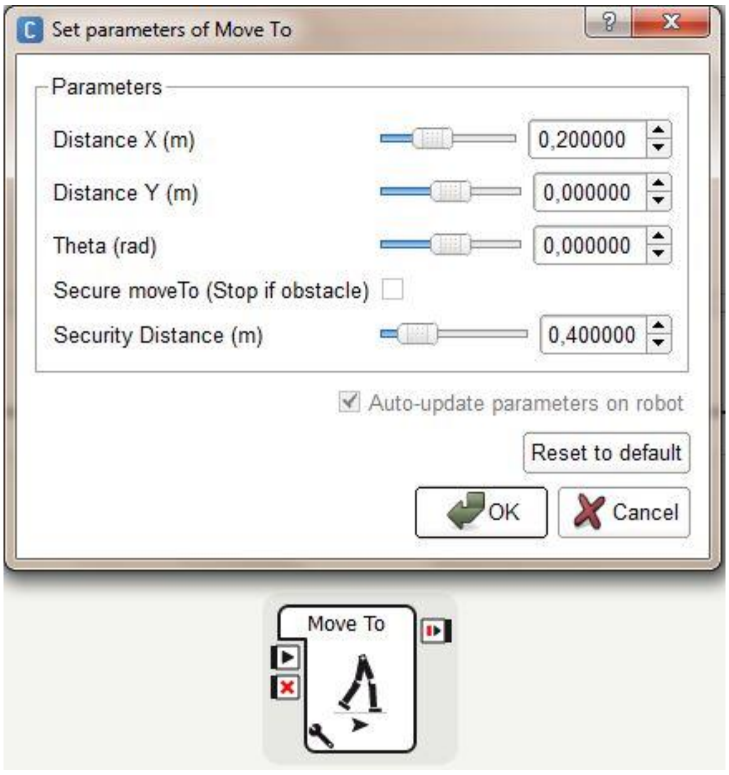


Figure 3.2 Box Library

Python gained more and more popularity, being widely used nowadays. It is a high-level programming language, supporting several paradigms (such as object-oriented, functional and procedural programming) and offers the possibility to express in fewer code lines.

The macro-blocks can be dragged and dropped from the Box Library to the working space – Flow Diagram panel, where they are represented by an icon, as it can be seen in figure 3.3. Some macro-blocks can have some input parameters that may be changed by the user as is the case of Move To function. For example, the Move To function makes the robot move on a certain direction over a certain distance; it gives the possibility setting the robot to move forward or backward (X or

Y) on a certain distance and at a given angle (Theta). It can be also be secured to stop if an obstacle appear during the movement at a certain Security distance.


Set parameters of Move To

Parameters

Distance X (m) 0,200000

Distance Y (m) 0,000000

Theta (rad) 0,000000

Secure moveTo (Stop if obstacle) ☐

Security Distance (m) 0,400000

☑ Auto-update parameters on robot

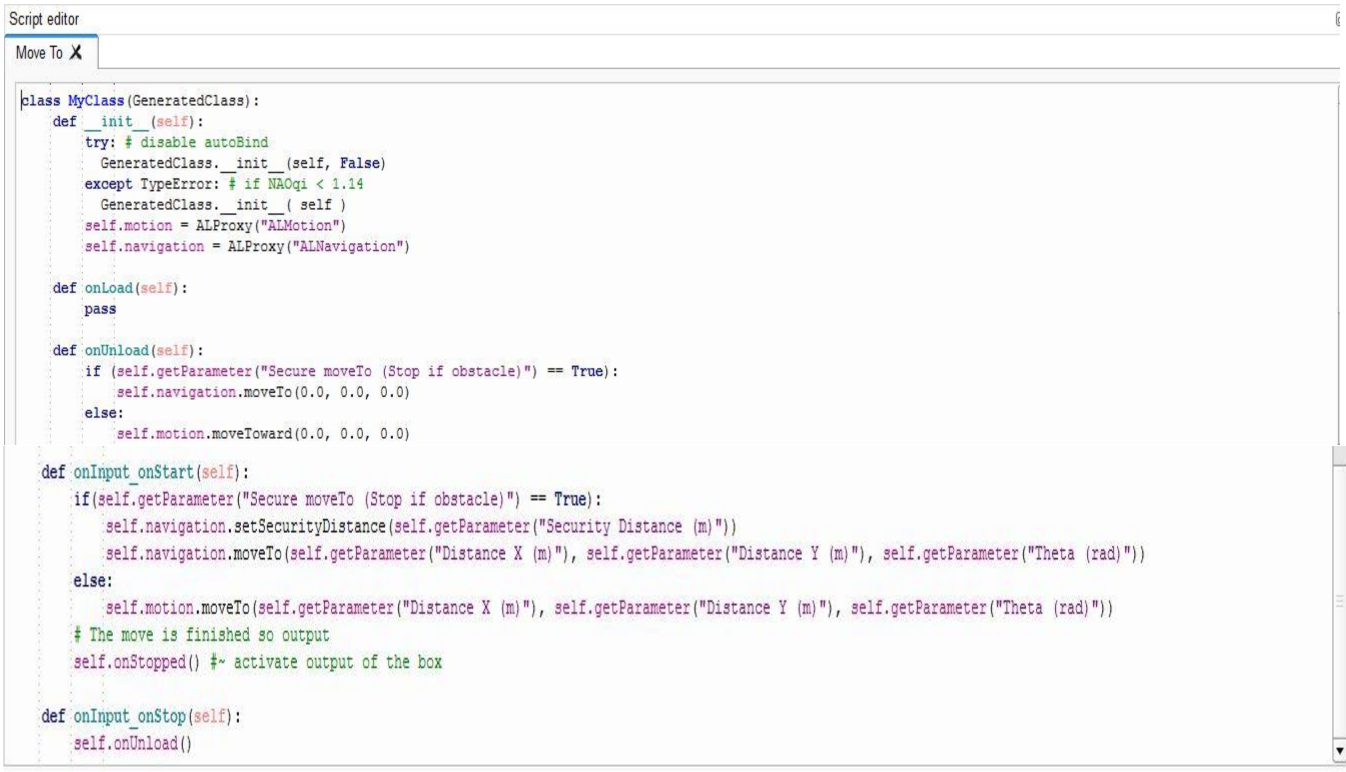
Figure 3.3 Move To macro-block – icon and parameters

Choregraphe is an open-source software suite, meaning the script of the function is available to the user, using the Script Editor. The user can modify the function in order to program NAO to perform the wanted action. Debugging is available when loading the behavior on the robot, in the Debug window.

Figure 3.4 represents the script behind the Move To macro-block; one class is defined MyClass and several methods are defined. The names of the methods are actually given by the template of the software; when trying to create a new box, from the beginning, the same methods appear. As it can be seen, this function uses AlMotion and AlNavigator modules and a very important feature must be highlighted: the UnLoad method is available in every function and represents unload of the behavior after it ended.

The macro-block or the Box is a fundamental object in Choregraphe and it can be very simple or very complex (multi-level Box). There are three types of boxes [18]:

- ✓ Script Box: it only includes a script.
- ✓ Flow Diagram Box: it includes a script and flow diagram (group of boxes linked with each other and linked with at least an input)
- ✓ Timeline Box: it includes a script and a timeline (allows enabling synchronization of the Boxes with the movements and the time, being based on a time ruler, composed of frames)



```
class MyClass(GeneratedClass):
    def __init__(self):
        try: # disable autoBind
          GeneratedClass.__init__(self, False)
        except TypeError: # if NAOqi < 1.14
          GeneratedClass.__init__( self )
        self.motion = ALProxy("ALMotion")
        self.navigation = ALProxy("ALNavigation")

    def onLoad(self):
        pass

    def onUnload(self):
        if (self.getParameter("Secure moveTo (Stop if obstacle)") == True):
            self.navigation.moveTo(0.0, 0.0, 0.0)
        else:
            self.motion.moveToward(0.0, 0.0, 0.0)
    def onInput_onStart(self):
        if(self.getParameter("Secure moveTo (Stop if obstacle)") == True):
            self.navigation.setSecurityDistance(self.getParameter("Security Distance (m)"))
            self.navigation.moveTo(self.getParameter("Distance X (m)"), self.getParameter("Distance Y (m)"), self.getParameter("Theta (rad)"))
        else:
            self.motion.moveTo(self.getParameter("Distance X (m)"), self.getParameter("Distance Y (m)"), self.getParameter("Theta (rad)"))
        # The move is finished so output
        self.onStopped() #~ activate output of the box

    def onInput_onStop(self):
        self.onUnload()
```

Figure 3.4 Move To – Script editor

3.3 NAO’s Behavior

The first plan on which the author of the thesis worked for the physical implementation of the paper was designing a behavior for the robot, having the required specifications. The behavior was developed using Choregraphe software and is found under the form of a project, a single compressed file with a CRG extension.

3.3.1 Deployment

The project (NAO behavior) developed using Choregraphe software program is called **AutomationSystemSpeechRecognition.crg** and it was implemented in order to have several functionalities: record the vocal command spoken by the user, send an e-mail to asrserverspeed@gmail.com with the acoustic signal file attached, wait while the .wav file in converted into text, fetch the e-mail account naoaldebaranrobot@gmail.com for the response e-mail, perform the action indicated by the vocal command.

The particularities of the entire flow made the choice of the Box to be a Timeline Box. The prototype of a Timeline Box can be found under the Template box library, or it can be created directly on the Flow Diagram panel, right click → Add a new box → set the Type as Timeline. The first step is ready, the Automation System Timeline Box is created. In order for the behavior to be functional, the Timeline Box must be connected to the output and to the input; drag the mouse from the onStart input of the behavior to the onStart input of the Timeline Box to create a link. Same mechanism is performed for the end part of the behavior (onStopped). Figure 3.5 displays the Flow Diagram panel with the Automation System Timeline Box linked to the start and the end.

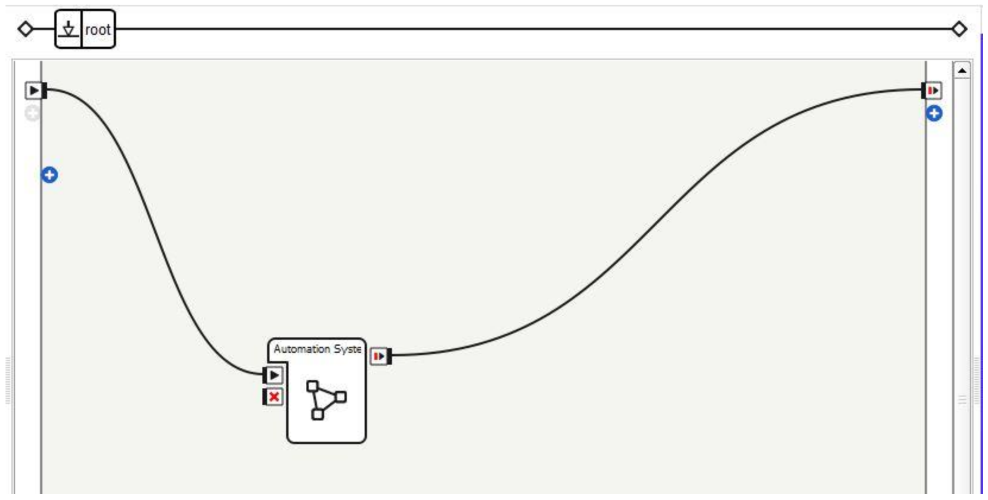


Figure 3.5 Automation System Timeline Box

Further, double-clicking the Timeline Box accesses the inner Timeline and the Timeline panel. Due to the particularities highlighted above, the Frame Ruler was divided into four parts. The Frame is the unit of the Timeline and each one has a number which corresponds to its position in the Timeline [19].

Figure 3.6 exhibits the Frame Ruler of Automation System; the Timeline consists of only one layer (SpeechRecognition) and the Frame Ruler is partitioned in four KeyFrames: Record, SendE, Wait, RetrieveE. Each KeyFrame represents a part of the behavior and can be composed of one or several macro-blocks, composing in fact four Flow Diagrams. The speed of the frame is one frame per second and it can be modified by the user. Each KeyFrame will be discussed separately.

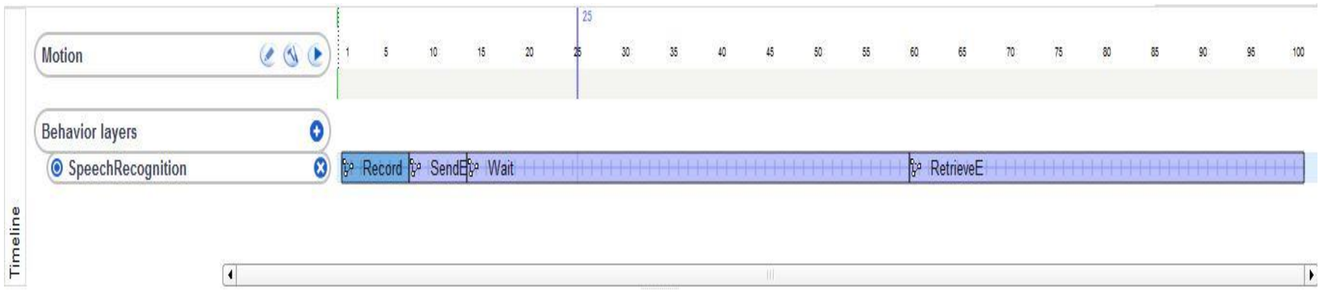


Figure 3.6 Automation System Frame Ruler

A. Record

Record represents the first KeyFrame of the Timeline and it lasts 8 seconds (frames). Its main role is to record the vocal command spoken by the user. A closer look at the Record shows what is displayed in figure 3.7.

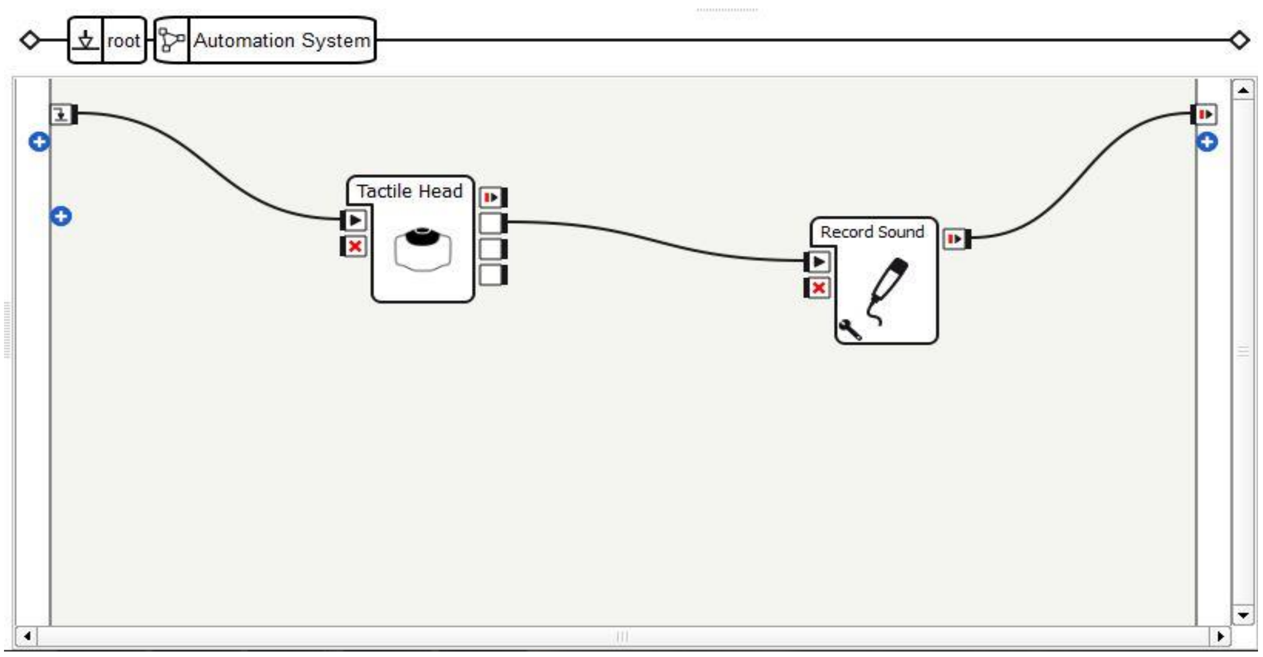


Figure 3.7 Record KeyFrame

The Record can be described as a Flow Diagram Box composed of two linked Boxes:

- ✓ Tactile Head – a Flow Diagram Box, acting as a trigger. It detects touch on the head tactile sensors (one of the three: front, middle, rear). In this particular behavior, the signal is sent farther if the front tactile sensor was touched. Actually, the robot starts recording only if the front of its head is touched.

- ✓ Record Sound – a Flow Diagram Box, composed of two Script Boxes: Get File Name and Record Sound File. The recorded file is a .wav file, named Record.wav, composed of signals coming from all four microphones NAO is being fitted to: front, sides and rear microphones. Due to the fact that the list of commands is formed of short commands, the record duration is set at 5 seconds.

The working principle of this KeyFrame is the following: the input signal activates the Tactile Head function, the Robot waits for the user to touch the front part of its head. When the sensor acquires the needed information (the front part of the head is touched), the signal is sent to the output of this function and becomes the input signal for Record Sound macro-block. From this moment, NAO records for 5 seconds (and the user starts speaking) and saves the file in the memory.

B. SendE

The second KeyFrame is the one that involves sending the e-mail to the specified recipient in order for the Java application running on the laptop to interrogate the account, fetch the new e-mail and save the attachment, namely the Record.wav audio file. SendE lasts 6 seconds and figure 3.8 displays the linked component Boxes:

- ✓ Send E-mail – a Script Box, in charge with sending the e-mail.
- ✓ Play Sound – a Flow Diagram Box; by accessing the inner Flow Diagram of the macro-block two instantiated Script Boxes are found: Get Attached File and Play Sound File. The purpose of this part is to let the user know that the e-mail was sent.

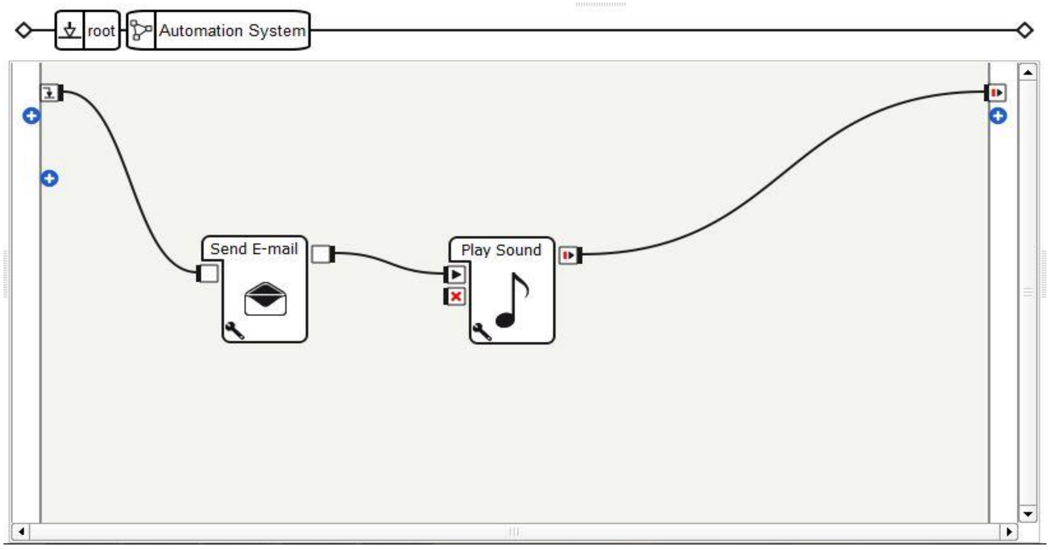
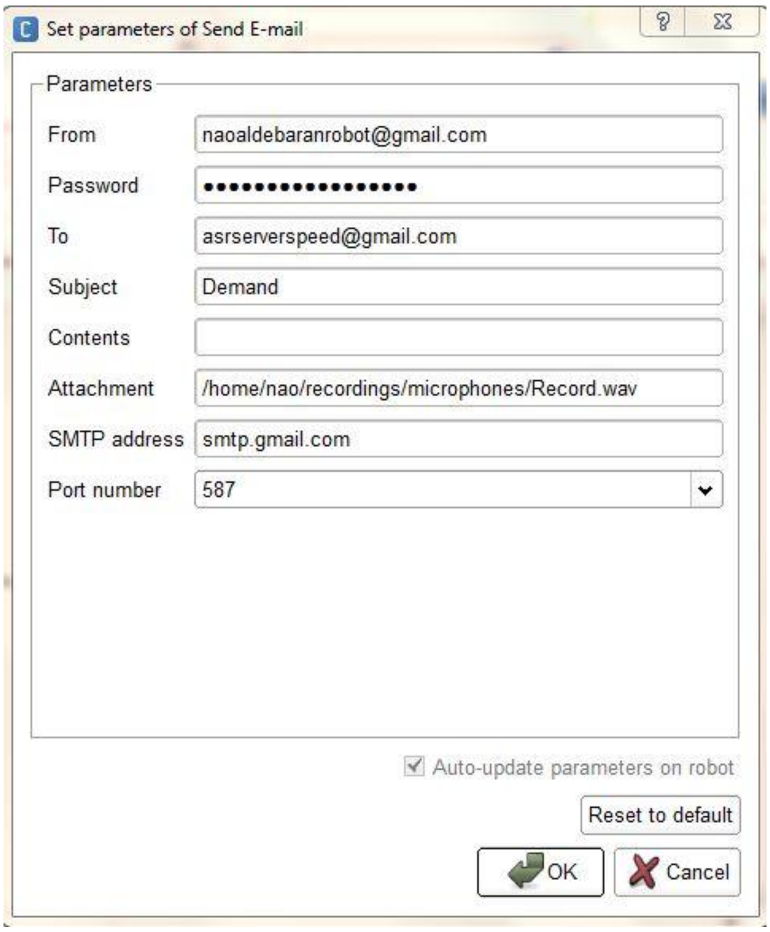


Figure 3.8 SendE KeyFrame

The working principle can be summarized as follows: after the Record KeyFrame is finished, the SendE KeyFrame is loaded. The onStart signal is received by the Send E-mail Script Box. The figure 3.9 reveals the parameters that were set in order for the e-mail to be sent properly.


Set parameters of Send E-mail

Parameters

From	naoaldebaranrobot@gmail.com
Password	•••••••••••••••••
To	asrserverspeed@gmail.com
Subject	Demand
Contents	
Attachment	/home/nao/recordings/microphones/Record.wav
SMTP address	smtp.gmail.com
Port number	587

☑ Auto-update parameters on robot

Figure 3.9 Send E-mail parameters

The protocol in charge with connecting to the server that hosts the e-mail service is SMTP (more details about it will be highlighted in Chapter 4), the address of the server being smtp.gmail.com and the port number 587. The e-mail is sent from naoaldebaranrobot@gmail.com, the recipient is asrserverspeed@gmail.com and the subject: Demand. NAO sends an empty e-mail, the important part being the attachment. One of the author contributions regarding this part of the application is represented by changing the box script of Send E-mail and insert the code line that sets the memory path where the Record.wav is saved: os.path = "/home/nao/ recordings/microphones/Record.wav".

After the e-mail is transmitted, NAO plays a sound, to announce this fact to the user. The onStopped signal of Send E-mail represents the onStart signal that activates Play Sound. The file to be played was loaded from the computer.

C. Wait

The speech-to-text transcription is time consuming, as well as the sending and fetching e-mails. For this waiting time, a third KeyFrame was introduced, namely Wait. Figure 3.10 exhibits the component macro-blocks. The Wait function is worth mentioning; it waits a certain time before sending a signal on the output. The waiting duration is a parameter set by the user and it depends on the time needed for the application MainApplication.java to run and send back the transcription to the robot. Tests were conducted at a peak hour for internet access, so the speed was low and the waiting time was up, the parameter was set at 46 seconds. Usually, this time is shorter (about 20 seconds). Figure 3.10 displays the construction of this particular frame

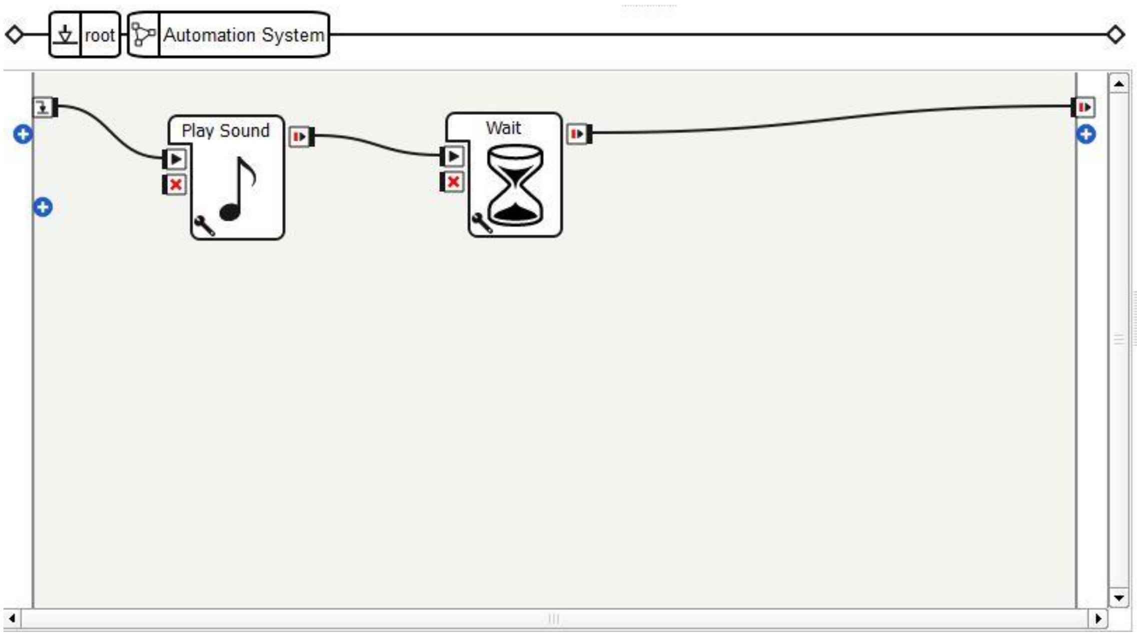


Figure 3.10 Wait KeyFrame

D. RetrieveE

The fourth KeyFrame is the complex part of the behavior. During this part, NAO fetches the e-mail containing the transcription of the vocal command and depending on the command, it performs the proper action:

- ✓ Stands up
- ✓ Walks one meter
- ✓ Sits down
- ✓ Moves the right hand
- ✓ Recognizes the person standing in its front.

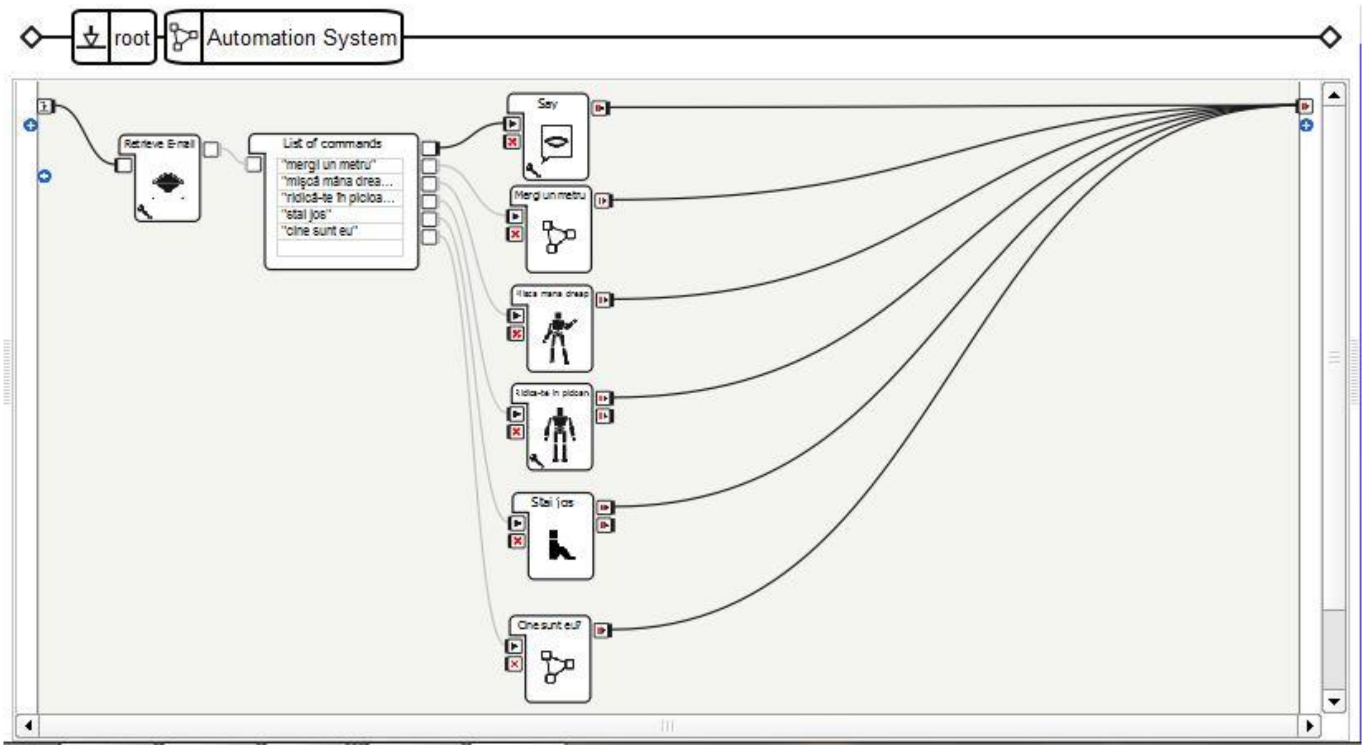
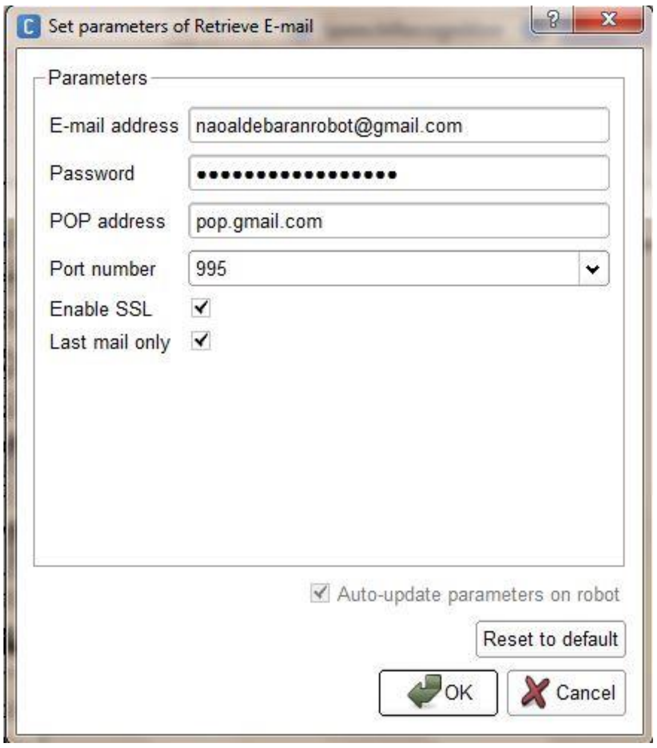


Figure 3.11 RetrieveE KeyFrame

✓ Retrieve E-mail: a Script Box that fetches the last e-mail from the server. The parameters to be set are available in figure 3.12.



Set parameters of Retrieve E-mail

Parameters

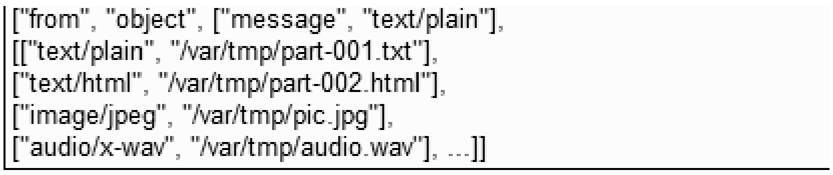
E-mail address	naoaldebaranrobot@gmail.com
Password	●●●●●●●●●●●●●●●●●●
POP address	pop.gmail.com
Port number	995
Enable SSL	✓
Last mail only	✓

✓ Auto-update parameters on robot

Figure 3.12 Retrieve E-mail parameters

For reading the e-mail, the protocol is POP3; the address for the server hosting the e-mail service is pop.gmail.com and the port number 995. The user can use between enabling and disabling SSL. The choice of the author was to enable SSL and retrieve only the last e-mail. Details of the account were set (E-mail address and Password).

In order to implement the entire list of commands, a Choice Script Box is used, namely List of Commands; in fact a case implemented with if – elif instructions. The input of this function is a String, while at the beginning, the output of Retrieve E-mail was an Array containing what figure 3.13 displays.



```
["from", "object", ["message", "text/plain"],
[["text/plain", "/var/tmp/part-001.txt"],
["text/html", "/var/tmp/part-002.html"],
["image/jpeg", "/var/tmp/pic.jpg"],
["audio/x-wav", "/var/tmp/audio.wav"], ...]]
```

Figure 3.13 Retrieve E-mail initial output

The author’s contribution can be highlighted in this section; besides setting the proper parameters, the two functions must be made compatible. This means changing the output from Array to String. The script of the Box was modified as the next paragraph shows:

```
nbMess = self.pop.stat()[0]
messageToRead = nbMess

if(self.getParameter("Last mail only")):
messageToRead = 1

restmp = []
for i in range( messageToRead ):
restmp = []
msg = "\n".join( self.pop.retr( nbMess - i )[1] )
mp = MailParser()
em = mp.parseMessage( msg )
content = mp.findMainText( em )
restmp.append(content[1])

res= content[1]
```

- ✓ List of Commands: a Switch Case Box which tests the input value and stimulates the output matching the input. If there is no matching output, the default output (onDefault) is stimulated.

1. “mergi un metru”

The first command is “walk a meter”; the Flow Diagram Box Mergi un metru is linked to the output corresponding to this input. The Box is a multilevel diagram and its content is shown in figure 3.14.

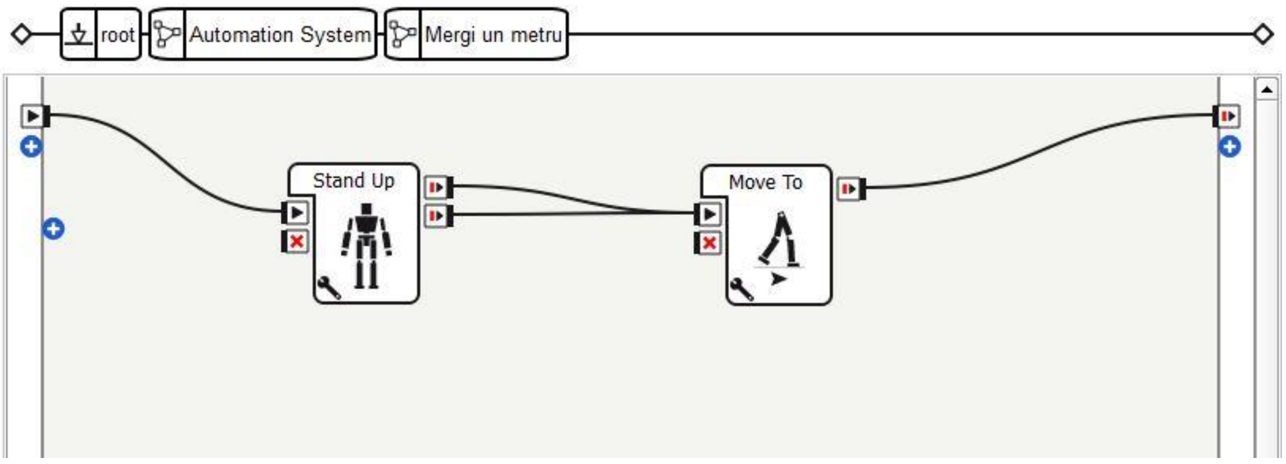


Figure 3.14Mergi un metru Box

Firstly it is checked if the robot is standing up; a Stand Up Box is linked to the input and its both outputs are further linked to a Move To Box. The two wires signify a successful and a failure action of standing up; if the robot is sitting down, it will first stand up and then it will walk a meter. Otherwise, if the robot is already standing up, the output signal will be transmitted to the failure output (NAO tries to stand up and considers the incapacity as a failure) and it walk a meter.

The Stand Up Box allows setting the number of tries before sending the output signal on the failure branch; it was set to 3 times.

2. “mişcă mâna dreaptă”

The second command is “move the right hand” and is implemented using a child Box as shown in figure 3.15. The input of this Script Box is linked to the output of the Switch Case Box corresponding to the input “mişcă mâna dreaptă”.

Figure 3.15 also shows the tree of the implemented behavior; the timeline Automation System has only one layer, named SpeechRecognition. Currently the navigation is done inside RetrieveE KeyFrame, inside the Misca mana dreapta Box.

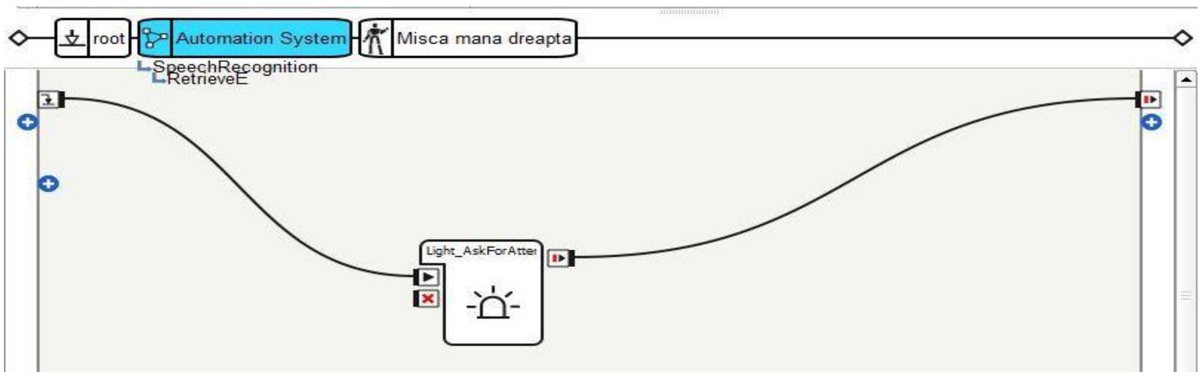


Figure 3.15 Misca mana dreapta Box

3. “ridică-te în picioare”

The third command is “stand up”; as for the two commands described before, the Script Box Ridica-te in picioare for this one is also linked to the output corresponding to the input “ridică-te în picioare”. The number of tries is set to 5 and the failure output for the Box is not taken into account. The child Box for the parent Box is GoTo Posture and is described in the next paragraph.

4. “stai jos”

The “sit down” command is similar to the third command; it is designed with the help of a Script Box as it can be observed in figure 3.16. The child Box is GoTo Posture; it permits the user to set the speed of the action as well as the type of the action NAO should perform. For this command, the name of the action is Sit and the speed is 80%.

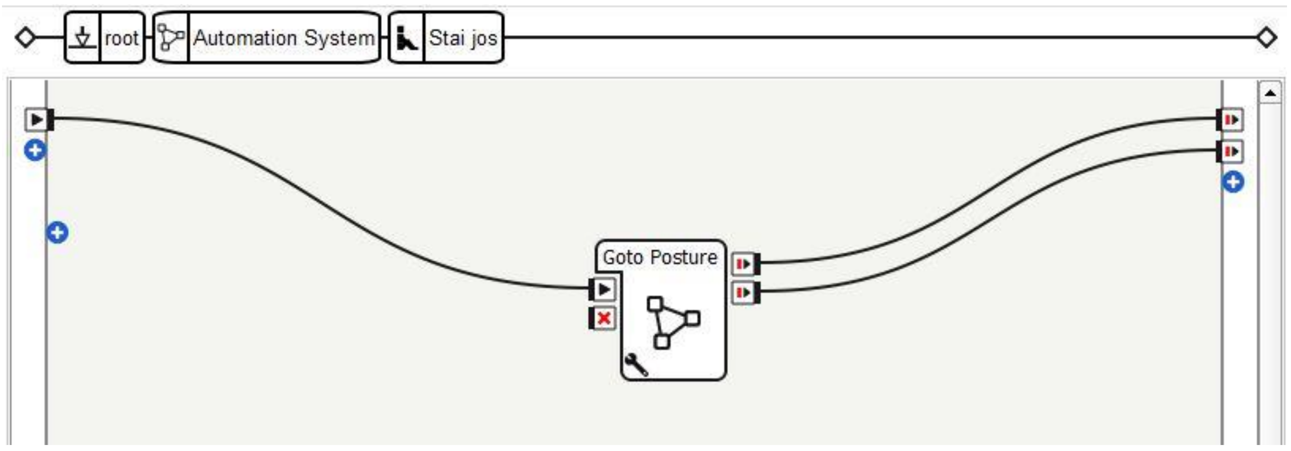


Figure 3.16 Stai jos Box

5. “cine sunt eu”

The last command represents a more complex command and is based on NAO ability to recognize images. In order to recognize a face, NAO must learn that particular face before. For this, the implementation of an additional behavior was necessary as figure 3.17 exhibits.

The input of the Learn Face function is a String representing the name of the face NAO must learn. The name is sent through e-mail (the modified version of Fetch E-mail function, as described above) and the learn process takes between 8 and 10 seconds.

Figure 3.18 displays the Flow Diagram Box named Cine sunt eu; it is composed of four Boxes divided in two separate cases. The first Box is Face Reco. Box having as output a String value, namely the name of the person NAO recognized or no name if NAO did not learn previously the face.

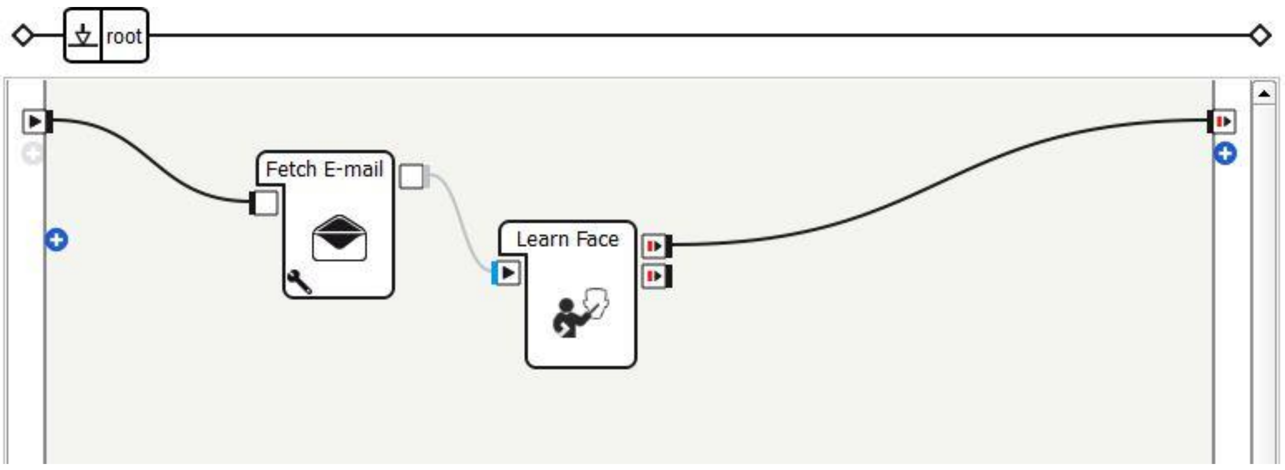


Figure 3.17 Learn Face behavior

The output of the Face Reco. Box is the input of a Switch Case with two possible actions. The first one, when the person standing in front of NAO is the author of this paper, meaning the output of the Box is "Diana", the recognition was successful so NAO will say the name of the person (Diana). The second one, when the person standing is not learned by the robot, it will say the next text "I do not recognize this person. I must learn the person before recognizing the face."

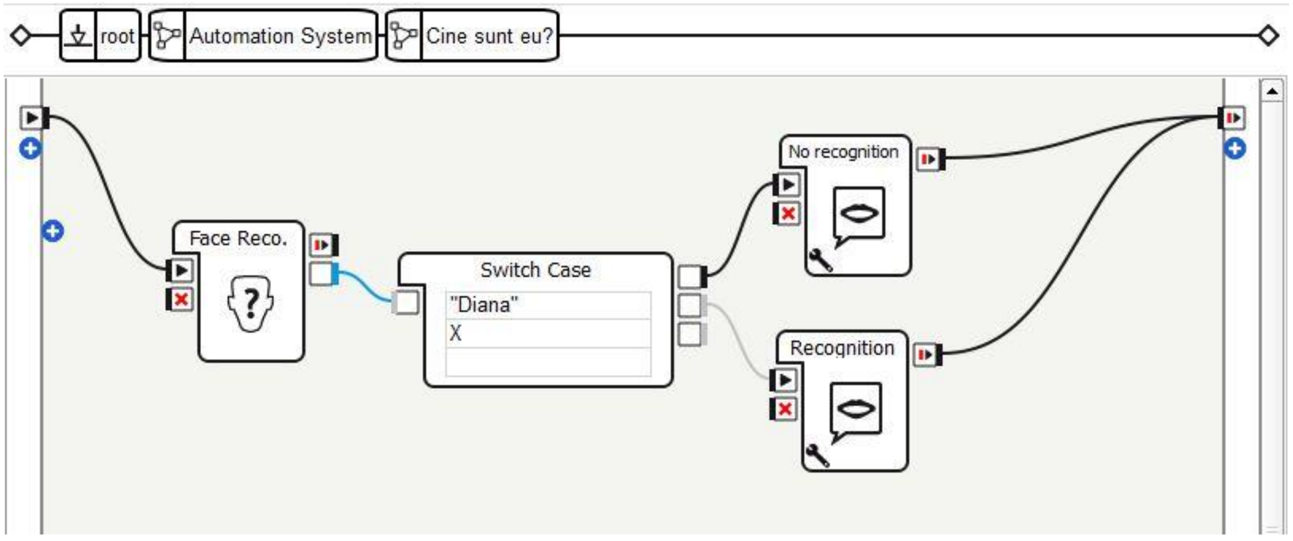


Figure 3.18 Cine sunt eu Box

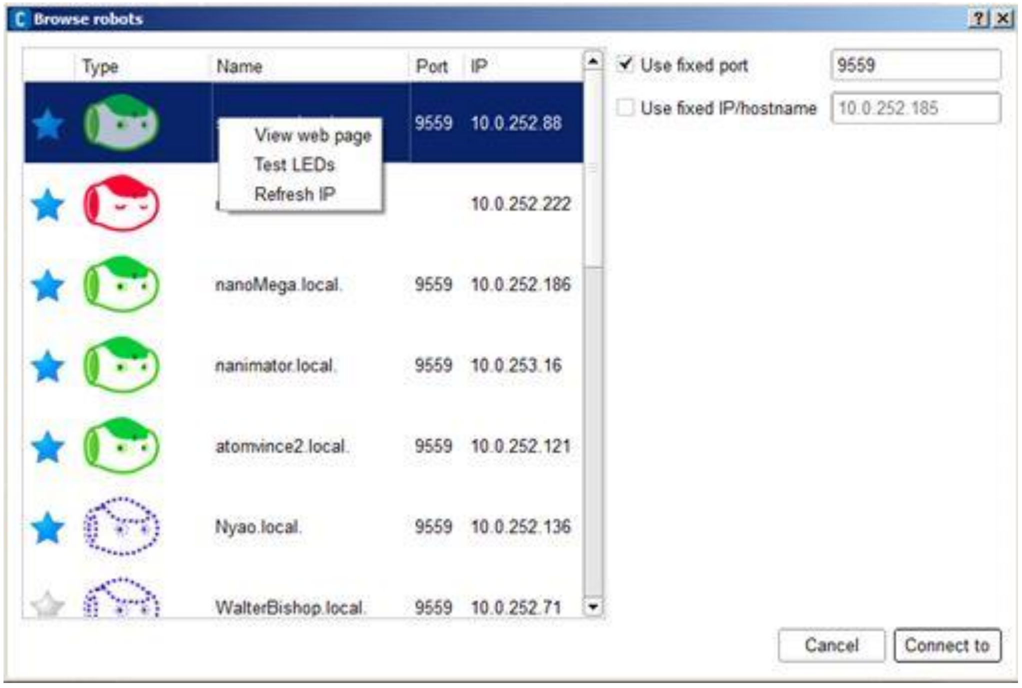
6. onDefault

There are situations when, because of the environmental noise or because other factors, the text-to-speech transcription is not done performed correct. In this situation, the content of the fetched e-mails does not reassembly any of the five commands. In this case, NAO will pronounce a text, helping the user in its further actions: "I could not understand what you said. Can you please reload and repeat?". For this, a Say Flow Diagram Box is used, formed of the localized text (the one from above) and the Say Text Box.

3.3.2 User Interface and Experience

Choregraphe itself represents a GUI software program; it is an intuitive and easy to work with program. The internet connection is very important, so firstly, a user must verify this aspect. The second phase is to obtain an IP address for NAO in the same LAN the computer is; for this a series of steps are presented in the documentation [19].

After launching Choregraphe, the user must connect to NAO using this software program. Under the tab Connection → Connect to, a window with all the available robots for connection is opened, as figure 3.19 displays:



Type	Name	Port	IP
		9559	10.0.252.88
			10.0.252.222
	nanoMega.local.	9559	10.0.252.186
	nanimator.local.	9559	10.0.253.16
	atomvince2.local.	9559	10.0.252.121
	Nyao.local.	9559	10.0.252.136
	WalterBishop.local.	9559	10.0.252.71

☑ Use fixed port 9559
☐ Use fixed IP/hostname 10.0.252.185

Figure 3.19 Choregraphe connecting to NAO [19]

Mainly, there are two possibilities to connect to a robot; the first one, identify it using its IP address and then double-click on its pictogram or, the second one, when for some reasons the software does not displays the robot pictogram, by directly typing the robot IP in the Use fixed IP/hostname tab. For all the robots, the connection port is 9559. Now, the user must press Connect to and if the connection was successful, a message appears.

The project must be opened first (File → Open project → AutomationSystemSpeechRecognition.crg) and after this, the Play icon must be pressed. At this moment, the behavior is successfully loaded onto the robot. The Frame Ruler starts to measure the seconds and a red bar shows the user the frame that is currently loaded.

Before loading the behavior onto the robot, the user must open NetBeans IDE 8.0 and run the Java project, as it is specified in paragraph 4.4.2. If the MainApplication.java is properly working and the behavior AutomationSystemSpeechRecognition.crg is successfully loaded, the user can start interacting with the robot.

In order to start recording, NAO's head must be touched on the front part; the user starts speaking the vocal command after this action and the robot records using the four microphones it has. In the next step, the attachment is sent via e-mail and this it is made known to the user by an audio sound played by the robot. NAO waits while the audio signal is processed by the server and the text transcription is sent back to it.

The last part of the behavior is the action itself; the e-mail is fetched and depending on the command uttered by the user, NAO will stand up, sit down, walk a meter, move the right hand or recognize the person standing in front of it.

CHAPTER 4

Client – Server Application

The second plan on which the author of the thesis worked for the physical implementation of the paper was designing and implementing an application able to make the link between the robot and the server on which the ASR is running. Since the main client (the robot) could not communicate directly with the server, a client–server architecture had to be developed, independently on the robot.

The protocol of communication between any client and the server running the ASR was previously developed and will be presented in this chapter. The protocol is based on sockets and .xml files transmission and a complex Java application was designed previously writing this paper [8].

The author contributions consist in implementing a Java application, using a set of protocols, with the following flow: access the e-mail server, fetch the e-mail message that was previously send by the robot and save the attachment (representing in fact the vocal command), send data to the server (using the existing protocol), receive the transcription of the vocal command (transmitted data) and send via e-mail plain text message the speech-to-text sequence back to the robot.

4.1 Client-Server Architecture and Communication Protocol

The application developed for the automatic speech recognition presented in this paper was written in Java programming language and follows the client-server architecture. The client-server model acts like a distributed application partitioning tasks between the server (service provider) and the client (service caller) [20].

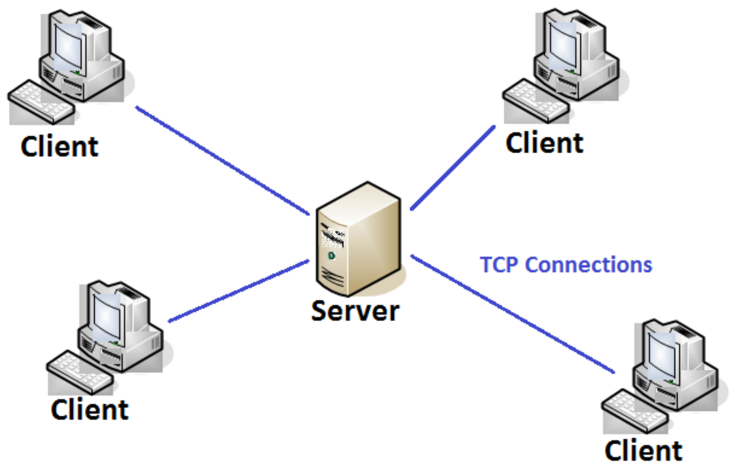


Figure 4.1 Client – Server architecture [20]

Figure 4.1 presents a simple client – server architecture; clients are represented by PCs and the connections between them and the server is through TCP connections. The client PCs represents in fact the interface between the users and the server; using them, the clients can request services and visualize responses received from the server. On the other hand, the server represents in fact a powerful computer running a series of specific programs, its main task involves waiting and responding to clients' demands. In our case, the server provides both access to the services and hardware resources used to process and transcript the vocal sound. The clients initialize connection to the server and request services. The client connects to the server through the internet.

Given the distributed structure of this kind of architecture, in order to communicate the client and the server, a connection must be established. For this, it is necessary that every application on the system to be uniquely identified at internet through an IP address and a port. Thus, each program is associated with a specific port on the server and each client that connects to that port can communicate with the server. For our application, the client connects to the server using the IP 141.85.252.230 and the port 5004.

Figure 4.2 presents the TCP/IP model, composed of 4 layers: Link, Internet, Transport and Application and in the next paragraph the way the client and the server communicate in the developed application will be presented, in terms of the protocols involved.

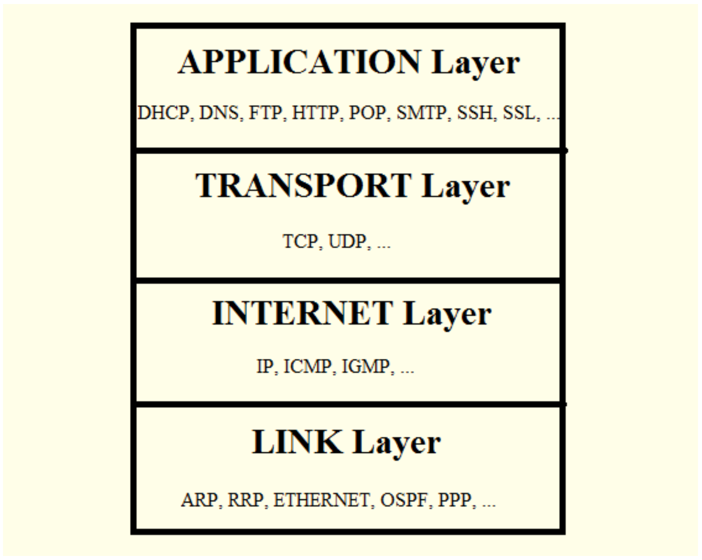


Figure 4.2 TCP/IP model

- ✓ **Application Layer** – the developed application uses at this layer a protocol previously designed [8]. The entire protocol can be summarized as a request – response protocol between the client and the server.

When trying to establish a connection with the server, the client must provide a socket address composed of IP address and the port (previously mentioned). After a successful connection, the client must authenticate using the username and the password.

```
<authenticateRequest username="diana.sandru" password="mewn$#11p"/>
```

After verifying the credentials, the server will respond with one of the two possible messages:

```
<authenticateResponse result="OK"/>
```

```
<authenticateResponse result="Failed"/>
<protocolErrorResponse description="authenticateResponse expected..."/>
```

If the authentication was successful, the client will interrogate the configurations that server supports `<getSupportedConfigurationsRequest/>` and the server will send a message with all the services supported by it `<getSupportedConfigurationsResponse>`.

For our application, the language is Romanian `<language name="Romanian">` and the domain identificator is 14 `<domain id="14">`. Information about the encoding and the frequency of the vocal data that can be sent to the server is provided:

```
<audioStreams encodings="PCM_SIGNED " audioFrequencyBands="narrow, wide"/>
```

Data must be encoded signed Pulse Code Modulation, a method to modulate the impulses in code, following the next flow: the analog audio stream is sampled and them it is quantized (regular – same number of bits for each sample or irregular – based on compression laws A and µ, in order to decrease the quantization noise). In our application, the data to be send to the server is stored into a WAV file, meaning it was encoded 16-bit signed PCM. The main advantage of the format is the fact that compression is not performed upon the audio stream; the quantization is regular (16 bits / sample) and none of the audio information is lost.

In the next step, the client requests a port for the stream transmission `<getAudioDataPortRequest/>`, and the server responds with the port number `<getAudioDataPortResponse port="500001"/>`. The client initiates a transcription request with all the options it needs and starts to send the acoustic signal received from the robot in the form of a stream of bytes.

The sever saves in the memory the received data, processes it and sends the answer back to the client in the form of a stream of bytes, representing the transcription of the acoustic signal, announcing in advance the client about the transcription start:

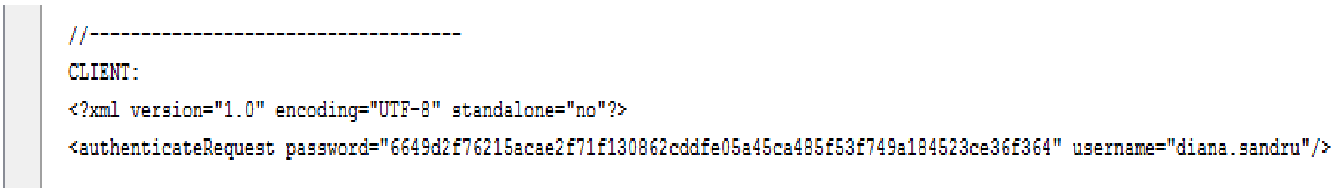
`<startTranscriptionAck/>`

The response includes the bestProcessedText, the bestRawText and also the duration of the initial stream along with information regarding the speaker: `<getTranscriptionResponse>`. When the transcription is completed, the server acknowledges the client `<stopTranscriptionAck/>`. The client closes the connection and an ACK response is received from the server.

- ✓ **Transport Layer** – the used protocol is TCP protocol; it segments the messages from the Application layer in smaller parts, called segments The main advantage of TCP (compared to UDP) is the fact that the received messages are reassembled at the destination in the same order they were send from the source and TCP guarantees that data was received.
- ✓ **Internet Layer** – the used protocol is the IP protocol; it takes the segments and encapsulates them in packets. The next step is to assign IP address and select the best route to the destination (client or server).
- ✓ **Link Layer**

Data exchange between the client and the server is transported using xml (XML), which represents a markup language, meaning a way to identify structures in a document. XML involves a set of rules for encoding the data in such way both human and machine can read it, being in fact a meta-language. XML was design to carry data and it doesn't have a predefined set of tags because it was conceived to structure and carry data.

Figure 4.3 displays an example of message exchanged between the client and the server, written in xml, version 1.0, encoded UTF-8 (each character is encoded with one to four octets). The markup is analyzed and the structured information is passed to the application by the XML parser.



```
//------------------------------------
CLIENT:
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<authenticateRequest password="6649d2f76215acae2f71f130862cddfe05a45ca485f53f749a184523ce36f364" username="diana.sandru"/>
```

Figure 4.3 Client – Server message

4.2 Java Mail

The application developed for the automatic speech recognition was written in Java programming language, class-based, object-oriented in order to have as few as possible interdependencies when programming [21]. The aim of the application was to fetch the e-mail message that was previously send by the robot and save the attachment (representing in fact the vocal command), send data to the server (using the existing protocol), receive the transcription of the vocal command (transmitted data) and send via e-mail plain text message the speech-to-text sequence (output of the ASR) back to the robot.

Working with e-mails made the author of the thesis use the JavaMail library. The JavaMail API provides a platform-independent and protocol-independent framework to build mail and messaging applications, providing facilities to send and read e-mails via SMTP, POP3 and IMAP. This framework represents a software frame on which changes are performed in order to achieve the desired functionalities.

The JavaMail reference implementation jar file used was javax.mail.jar, that also includes the SMTP, IMAP and POP3 protocol providers. The protocols used for this application will be discussed in the next subchapter. The JavaMail API was designed to satisfy the following development and runtime requirements [22]:

- ✓ Lightweight classes and interfaces make it easy to add basic mail-handling tasks to any application.
- ✓ Supports the development of robust mail-enabled applications.

In figure 4.4 it is presented the mail-enabled part of the entire application. The jar file javax.mail.jar was used along with the JDK 1.7 found in NetBeans IDE 8.0, the open source application development medium. NetBeans is used to interact with the user and display message content and some verification messages during the running.

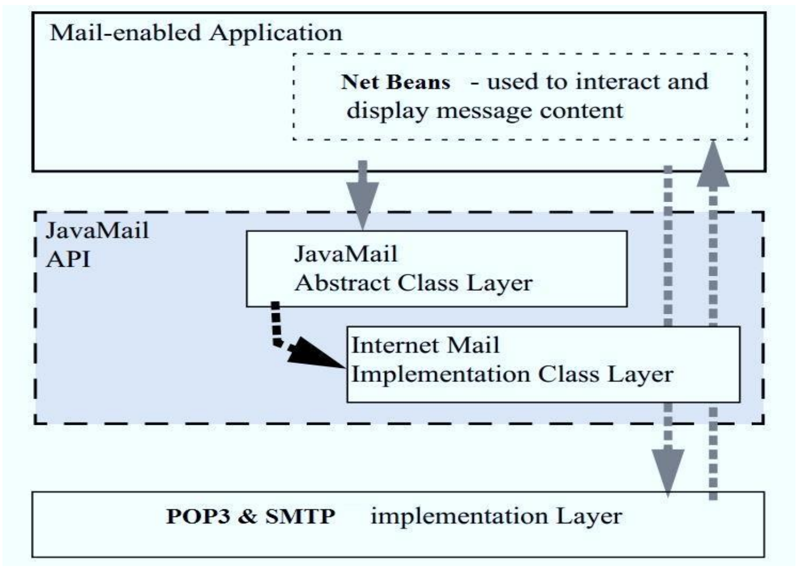

Mail-enabled Application
Net Beans - used to interact and display message content
JavaMail API
JavaMail Abstract Class Layer
Internet Mail Implementation Class Layer
POP3 & SMTP implementation Layer

Figure 4.4 Mail-enabled part of the application

4.3 The Server – Language, Phonetic and Acoustic Resources

The client – server architecture implies two instances: the client and the server. While the client instance was developed entirely by the author of the paper, the services offered by the server were previously developed by Speech and Dialogue Research Laboratory. Though, the resources necessary for the ASR construction must be presented.

Because the robot has to react to a limited set of five orders, formed of maximum three words each, the resources for the service the server must offer were limited; a control – command recognition task was implemented:

- ✓ **The Language model**

The text corpus for this application is not a wide corpus, being composed of only 13 words: “cine”, “sunt”, “eu”, “ridică”, “mâna”, “dreaptă”, “mergi”, “un”, “metru”, “stai”, “jos”, “ridică-te”, “în”, “picioare”. Due to the fact that the vocabulary for this recognition tasks has a well-defined succession order for the words, a Finite State Grammar language model is used.

A FSR is a graph model in which nodes represent words of vocabulary recognition task and the arcs of the graph represent the transitions between words. Such language model explicitly specifies all sequences of words allowed by the grammar recognition task. Moreover, to each arc may be assigned a cost, indicating the likelihood that a word is preceded by another.

- ✓ **The Acoustic model**

The acoustic model was created with the utilitarian Sphinxtrain included in CMUSphinx. HMMs with five states (from which three emissive) were used in order to model the phones; phones depend on the context.

- ✓ **The Phonetic model**

The phonetic dictionary is larger than 13; it contains almost 64000 distinct words and 96654 pronunciation forms, for all the language models existing on the server and it was created previously, by the Speech and Dialogue Research Laboratory.

The server used for this project has the following characteristics:

- ✓ Processor: Quad-Core Intel® Xeon®
- ✓ 16 GB RAM,

while the server application is the part of the architecture that processes the acoustic signal received from the client and returns the transcription. The access to the server implies creating an independent execution wire for each client. The server runs in command line and accepts connection on TCP port 5004. Each client that wants services transmits the vocal signal under the form of bytes stream using the socket allocated by the server.

4.4 The Client Application

The client part of the application was developed by the author of the paper using design specification of the JavaMail framework, offered by Oracle [23] and integrating the communication protocol with the server established previously.

4.4.1 Implementation

The application developed using NetBeans IDE 8.0 (JDK 1.7) software program is called **MainApplication.java** [the code can be found in Appendix 2] and it was implemented in three steps:

- ✓ First step was implementing an application able to interrogate periodically the e-mail, fetch new e-mails and download the attachments (if existing) in a predefined memory path;
- ✓ Second step represented the communication with the server; using the existing protocol and application (TranscriberClient.java), the contribution of the author was rethinking the application developed by [8] in order to obtain what was useful for MainApplication.java [the code can be found in Appendix 3];
- ✓ Third step was adding the application the feature that enables it to send via e-mail to the robot the transcription received from the server, in order the command to be further fetched by the robot.

In order to obtain the wanted functionalities, a series of packages (java.io, javax.mail, java.util, javax.mail.internet, javax.xml) and classes were included in the project.

The java.io package is primarily focused on input and output to files and network streams.

```
import java.io.File;
import java.io.IOException;
```

However, this package does not contain classes to open network sockets which are necessary for network communication. Once you have opened a socket (network connection) though, one can read and write data to and from it java.io's `InputStream` and `OutputStream` classes. This is the reason why in TrascriberClient, the entire package `import java.io.*` is included together with `import java.net.Socket; import java.nio.file.*;` and further more inherited by MainApplication.

```
import java.security.NoSuchAlgorithmException;
```

A series of utilities are also used in order to set the properties (in order to be able to read the e-mail encrypted) or to set the date of the sent e-mail.

```
import java.util.Properties;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.Date;
```

The choice of every class from javax.mail package will be explained later, during the application presentation.

```
import javax.mail.Authenticator;
import javax.mail.Address;
import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Multipart;
import javax.mail.NoSuchProviderException;
import javax.mail.PasswordAuthentication;
import javax.mail.Part;
import javax.mail.Session;
import javax.mail.Store;
import javax.mail.Transport;
import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeBodyPart;
```

Additional to parser inherited from TranscriberClient application, an exception class was generated automatically, along with the Transformer Exception, a class that specifies an exceptional condition that may appear during the transformation process.

```
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.TransformerException;

import org.xml.sax.SAXException;
```

The following diagram (figure 4.5) exhibits the attributes and operations of MainApplication and also the link between it and TranscriberClient. The attributes of MainApplication are the saveDirectory – used to indicate the memory path were the attachment is saved and the two parameters that compose the socket used to connect to the server, liveTranscriberServerAddress – IP address and liveTranscriberServerPort – port number. MainApplication is composed of three operations (setSaveDirectory, mainLoop, sendEmail) along with the main function.

First operation from MainApplication sets the directory where the attached files will be stored; it is a single parameter function – dir is the memory path of the directory.

```
    public void setSaveDirectory(String dir) {
        this.saveDirectory = dir;
    }
```

The first phase of the implementation was developing a loop to perform the polling action, interrogating periodically the e-mail account for new e-mails, at intervals of one second `Thread.sleep(1000)`. This parameter can be modified by the user. The void function mainLoop was created, having as parameters the one necessary to configure the account asrserverspeed@gmail.com, the account were the vocal command spoken to the robot is sent to, such as: host server for the account, port number, the user name and the password.

```
public void mainLoop(String host, String port,
                    String userName, String password)
```

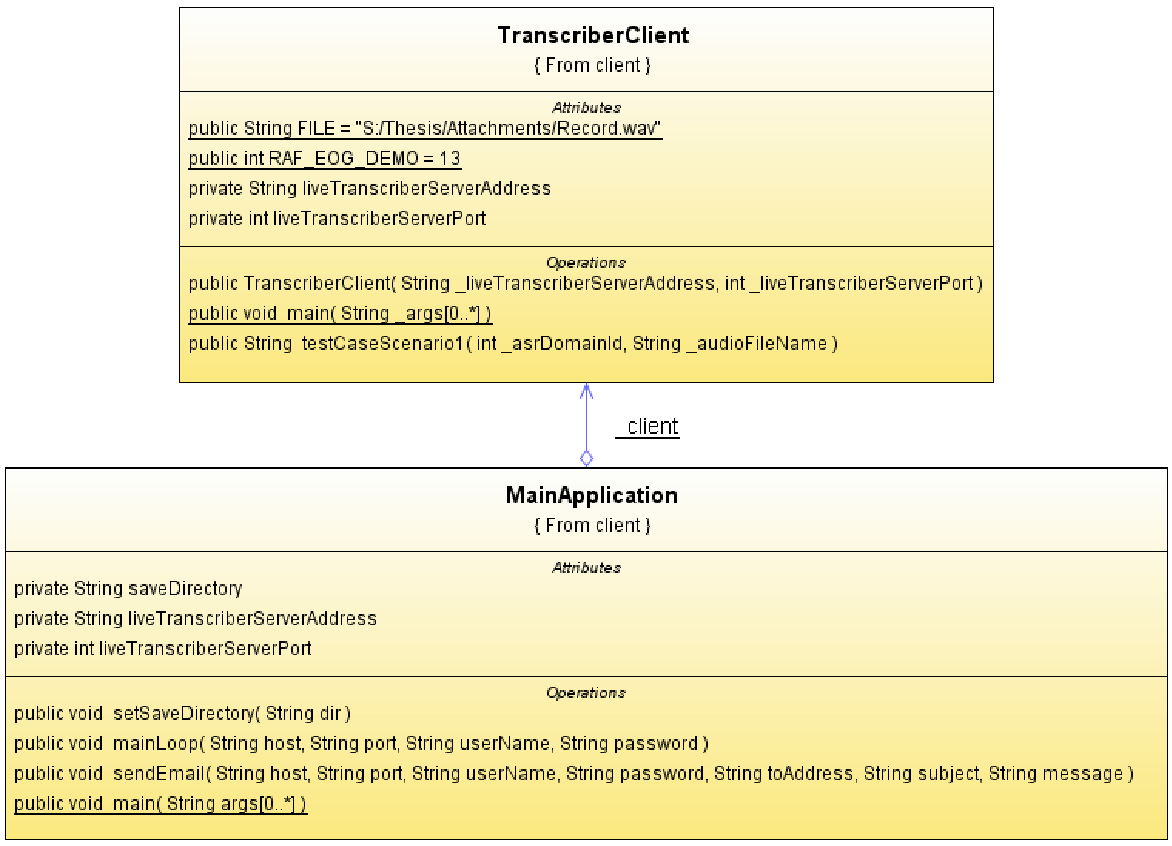

TranscriberClient
{ From client }
Attributes
public String FILE = "S:/Thesis/Attachments/Record.wav"
public int RAF_EOG_DEMO = 13
private String liveTranscriberServerAddress
private int liveTranscriberServerPort
Operations
public TranscriberClient(String _liveTranscriberServerAddress, int _liveTranscriberServerPort)
public void main(String _args[0..*])
public String testCaseScenario1 (int _asrDomainId, String _audioFileName)
client
MainApplication
{ From client }
Attributes
private String saveDirectory
private String liveTranscriberServerAddress
private int liveTranscriberServerPort
Operations
public void setSaveDirectory(String dir)
public void mainLoop(String host, String port, String userName, String password)
public void sendEmail(String host, String port, String userName, String password, String toAddress, String subject, String message)
public void main(String args[0..*])

Figure 4.5 MainApplication and TranscriberClient link

In order to be able to connect to the server, properties of the object had to be added dynamically. Thus, properties object has been created as an object of java.util.Properties library.

For the connection to the server, POP3 protocol was chosen between it and IMAP. The choice was performed based on the two Internet standard protocols to retrieve e-mails characteristics:

- ✓ POP3 : Post Office Protocol version 3 - downloads all e-mails from the server and stores them on the computer. The e-mails are removed from the server and only stored locally in the e-mail client program.
- ✓ IMAP : Internet Message Access Protocol - syncs the e-mail client program with the server. E-mails continue to stay on the server and they can be seen from many e-mail client programs or devices.

Because the application was developed to continuously interrogate the e-mail for new commands, download the attachment and send to the server for transcription, POP3 was used in order not to send the same attachment over and over again; the fetched e-mail must be marked as read. For Gmail, the server address is `pop.gmail.com` and the port number `995`.

The following code lines describe the POP3 server, by initializing the following:

`properties.put("mail.pop3.host", host);` - The POP3 server to connect to.

`properties.put("mail.pop3.port", port);` - The POP3 server port to connect to, because the connect() method doesn't explicitly specify one.

Also, the disable top property is set true, The POP3 server port to connect to, if the connect() method doesn't explicitly specify one.

```
properties.put("mail.pop3disabletop", "true");
```

Further, the SSL settings of the connection are set (in order to provide protection to the connection): the POP3 socket is created, specify the port to connect when the socket factory is used and also set the mail.pop3.socketFactory.fallback to false, meaning even if the creation of the socket will fail using javax.net.SocketFactory class, it must be retried using the same class.

```
properties.setProperty("mail.pop3.socketFactory.class",
        "javax.net.ssl.SSLSocketFactory");
properties.setProperty("mail.pop3.socketFactory.fallback", "false");
properties.setProperty("mail.pop3.socketFactory.port",
        String.valueOf(port));
```

After the settings have been done, a session object must be created in order for the application to connect to the POP3 store. The Session class represents a mail session and is not sub-classed. It collects together properties and defaults used by the mail API's. The getDefaultInstance method is used to create a Session object, and the parameters are the Authenticator and the Properties. The Authenticator is set to null, letting anyone to create a Session object, and the Properties are the one initialized above.

```
Session session = Session.getDefaultInstance(properties, null);
```

The polling is set using an infinite loop, a `while (true)` instruction. First step in connecting to the message store; Session class acts as a factory for Store and Transport objects that implement specific access and transport protocols. So, the Session object’s getStore method is used to connect to the default POP3 store. This method returns a Store object subclass that supports the POP3 access protocol.

```
Store store = session.getStore("pop3");
store.connect(host, userName, password);
```

If the connection is successful, the Inbox folder from the Store is opened (reference to it is obtained), and then Message objects are fetched (as arrays, using the getMessages() method). The Store class defines a database that holds a Folder hierarchy and the messages within. The Store also defines the access protocol used to access folders and retrieve messages from folders. The Folder class represents a folder containing messages.

```
Folder folderInbox = store.getFolder("INBOX");
folderInbox.open(Folder.READ_ONLY);

Message[] arrayMessages = folderInbox.getMessages();
```

Although the useful part from the e-mail is the attachment, optionally we can display the details regarding the sender, the subject, the date, the address of the sender. For that, several methods are used to extract from the message array (which is in fact the retrieved e-mail), the needed information: getFrom(), getSubject(), getSentDate(), getContent(). Figure 4.6 displays the results returned by each method and what information can be extracted from it.

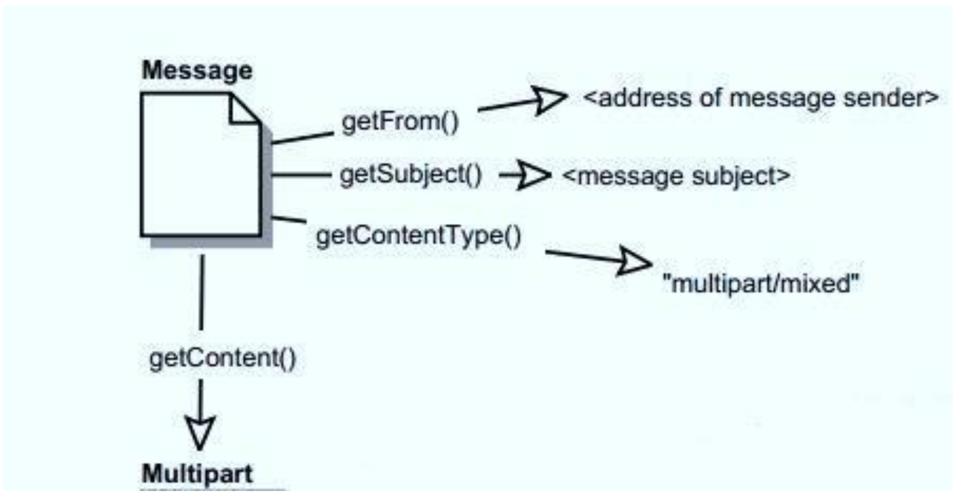


Figure 4.6 E-mail message construction

Using an If instruction it will be checked if the message contains any attachments. If it does, the attachments are extracted, but not before counting their number, and then saved in the memory at the place indicated by the setSaveDirectory method.

```
      if (contentType.contains("multipart")) {
                  Multipart multiPart = (Multipart) message.getContent();
                  int numberOfParts = multiPart.getCount();
                  for (int partCount = 0; partCount < numberOfParts;
partCount++) {
                        MimeBodyPart part = (MimeBodyPart)
multiPart.getBodyPart(partCount);
                        if
(Part.ATTACHMENT.equalsIgnoreCase(part.getDisposition())) {
                              String fileName = part.getFileName();
                              part.saveFile(saveDirectory + File.separator +
fileName);
```

In order to obtain the attachment, we define an object part, related to the MimeBodypart class from javax.mail.internet package. MIME, namely Multipurpose Internet Mail Extensions, represents a standard aimed to extend the format of e-mails to support non-text attachments, message body with more parts. This class represents a MIME body part. It implements the BodyPart abstract class and the MimePart interface. MimeBodyParts are contained in MimeMultipart objects. MimeBodyPart uses the InternetHeaders class to parse and store the headers of that body part.

The extraction of the attachment is performed only if the disposition is true; `Part.ATTACHMENT` shows the fact that this part must be presented as an attachment, leading to the next statement: If the disposition for the part is to be presented as an attachment, it will be saved to the memory.

The first step is now complete, the account was interrogated, the e-mail was retrieved and the attachment was saved. The next step represents the communication with the server and is implemented using the application TranscriberClient inside MainApplication. For this, a static reference is defined as link to the application `private static TranscriberClient _client;`

Actually, a new object TranscriberClient is defined in the main function, using the attributes `liveTranscriberServerAddress, liveTranscriberServerPort`

```
        _client = new TranscriberClient(liveTranscriberServerAddress,
liveTranscriberServerPort);
```

Using a Try instruction, the server is called with the extracted attachment (using the reference to the application and the method testCaseScenario1); if the entire protocol described in section 4.1 is functional, the response from the server is the transcription of the vocal command, which actually represents the text of the e-mail that will be sent back to the robot.

```
        mailMessage = _client.testCaseScenario1(14,saveDirectory + File.separator
+ fileName);
```

The last step is to send an e-mail to the address naoaldebaranrobot@gmail.com, account that is interrogated by the robot, containing as plain text the transcription from the server. Inside a Try instruction, the method sendEmail is called, along with the needed parameters. For e-mail sending, the protocol used is SMTP, namely Simple Mail Transfer Protocol, a protocol that for Gmail has the following attributes: server `smtp.gmail.com` and port `587`.

```
        sendEmail(host2, port2, mailFrom, password2, mailTo,
                                        subject2, mailMessage);
```

In a similar way to the settings applied for the POP3 server, a properties object is created for SMTP and the SMTP port and server to connect to are set:

```
          properties.put("mail.smtp.host", host);
          properties.put("mail.smtp.port", port);
```

Also, other two properties are set true: mail.smtp.auth and mail.smtp.starttls.enable. The first property attempts to authenticate the user using AUTH command, and the second one enables the use of the `STARTTLS` command to switch the connection to a TLS (SSL)-protected connection before issuing any login commands.

```
          properties.put("mail.smtp.auth", "true");
          properties.put("mail.smtp.starttls.enable", "true");
```

This time, the session is created using the Authenticator class, representing an object that knows how to obtain authentication for a network connection. In order to use this class, a subclass is created, and an instance of that subclass in registered with the session when it is created. When authentication is required, the system will invoke a method on the subclass (getPasswordAuthentication).

```
        Authenticator auth = new Authenticator() {
        public PasswordAuthentication getPasswordAuthentication() {
            return new PasswordAuthentication(userName, password);
        }
    };

          Session session = Session.getInstance(properties, auth);
```

The new e-mail is created using the MimeMessage class, a class that represents a MIME style email message. In order to create new MIME style messages, an empty MimeMessage object is instantiated (MimeMessage class provides a default constructor that creates an empty MimeMessage object) and then it is filled with appropriate attributes and content later by invoking the parse methods: setFrom, setRecipients, setSubject, setSentDate, setText. Date is set using the Date utility class, and the Recipients are set using an array of addresses.

```
    Message msg = new MimeMessage(session);

    msg.setFrom(new InternetAddress(userName));
    InternetAddress[] toAddresses = { new InternetAddress(toAddress) };
    msg.setRecipients(Message.RecipientType.TO, toAddresses);
    msg.setSubject(subject);
    msg.setSentDate(new Date());
    msg.setText(message);
```

In order to transmit the e-mail, the class Transport and the send method are used. In the end the session and the Inbox folder are closed.

```
    Transport.send(msg);
```

The main function is void, public and static and few initializations and method calls are performed. The IP and the port for the server connection are set (in order to form the socket), the `_client` is created and the directory for saving the attachment is set. An object from MainApplication class is created and then the mainLoop method is called with the characteristic parameters.

The TranscriberClient application is composed of two operations along with the main function. The main class is TranscriberClient with four attributes (server's IP and port, file to be sent name and the number of the domain).

In the main function, an object of the class is created and the testCaseScenario1 method is called with the previously set parameters.

```
   public static void main(String _args[]) throws Exception {
     String _liveTranscriberServerAddress = "141.85.252.230";
     int _liveTranscriberServerPort = 5004;

     TranscriberClient _client = new
TranscriberClient(_liveTranscriberServerAddress, _liveTranscriberServerPort);

    String _response = _client.testCaseScenario1(DOMAIN, FILE);
 }
```

The testCaseScenario1 is String type and return the transcription of the vocal stream.

```
_response = _transcription.getBestRawText();
```

The contribution of the author of this paper with respect to this part of the application can be summed up as follows:

- ✓ Understand the communication protocol and then simplify it to fit the demands of MainApplication; this implied the reduction of the communication protocol only to Romanian language (English and Albanian were left apart) and to domain number 14 (the interest domain for the application).
- ✓ Pick the best scenario from four existing ones, the one that fitted better the MainApplication.java – namely testCaseScenario1
- ✓ Change the type of the method testCaseScenario1 from void to String, such that the client declared in MainApplication to return exactly the transcription of the vocal command
- ✓ Determine that the best response received from the server is

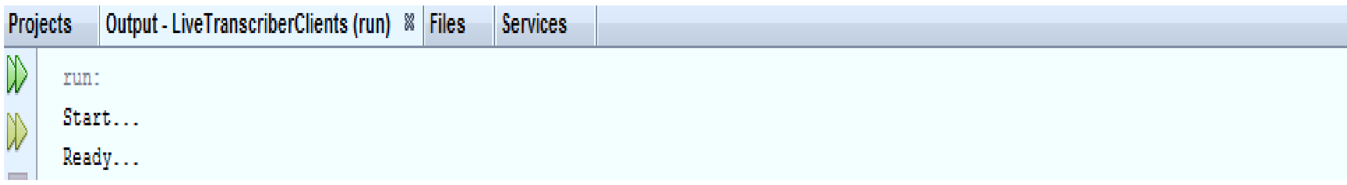
```
_response = _transcription.getBestRawText();
```

4.4.2 User Experience

The application is not accompanied by a Graphics User Interface because the main output of the entire system is represented by the robot and the actions it performs as a result of the speech-to-text transcription. However, a series of messages were printed on the screen during the running of the application such that the user to be able to see the current status of each phase.

The application MainApplication.java must be running when the behavior is loaded onto the robot. For that, few steps must be followed by the user: check if the connection to the internet is open NetBeans IDE 8.0 (an older version may cause problems with the application), load the entire project LiveTranscriberClients, search for the MainApplication.java under the Source Packages → org.etti.speech.transcriber.client and press the Run File icon for it. After loading the necessary resources, an Output panel will appear and this will be the place where the messages will be printed.

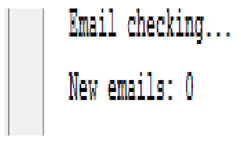
The first phase in the main function is creating the client object for TranscriberClient; after performing this task, a message will be printed on the screen as figure 4.7 displays:



```
run:
Start...
Ready...
```

Figure 4.7 Start of the application and connect to the POP3 server

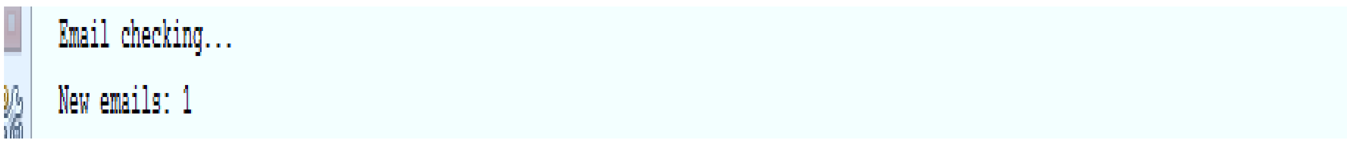
After the connection to the POP3 server was successful, a „Ready...” message is displayed. The next phase represents the Inbox folder checking for new messages. The step is presented to the user with the help of a message „Email checking...”.



```
Email checking...
New emails: 0
```

Figure 4.8 Application checks new e-mails

The polling action is performed, as figure 4.8 displays, and the application interrogates the account at 1 second in order to detect the e-mail sent by the robot, having attached the vocal command that has to be transmitted.



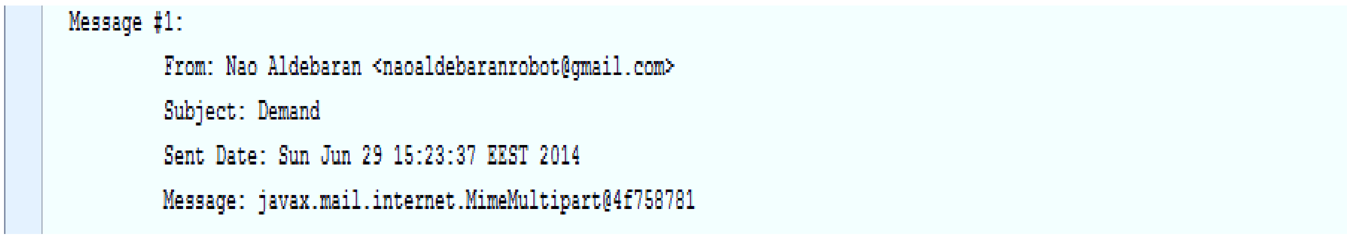
```
Email checking...
New emails: 1
```

Figure 4.9 Application detects a new e-mail

While the polling is done, the robot records the command, attaches the .wav file to the e-mail and sends it to the account that is periodically interrogated by the application. When a new e-mail is detected, the user is announced, as figure 4.9 shows.

Before initializing the connection to the server, another message is displayed; it contains the message details, as it can be seen in figure 4.10:

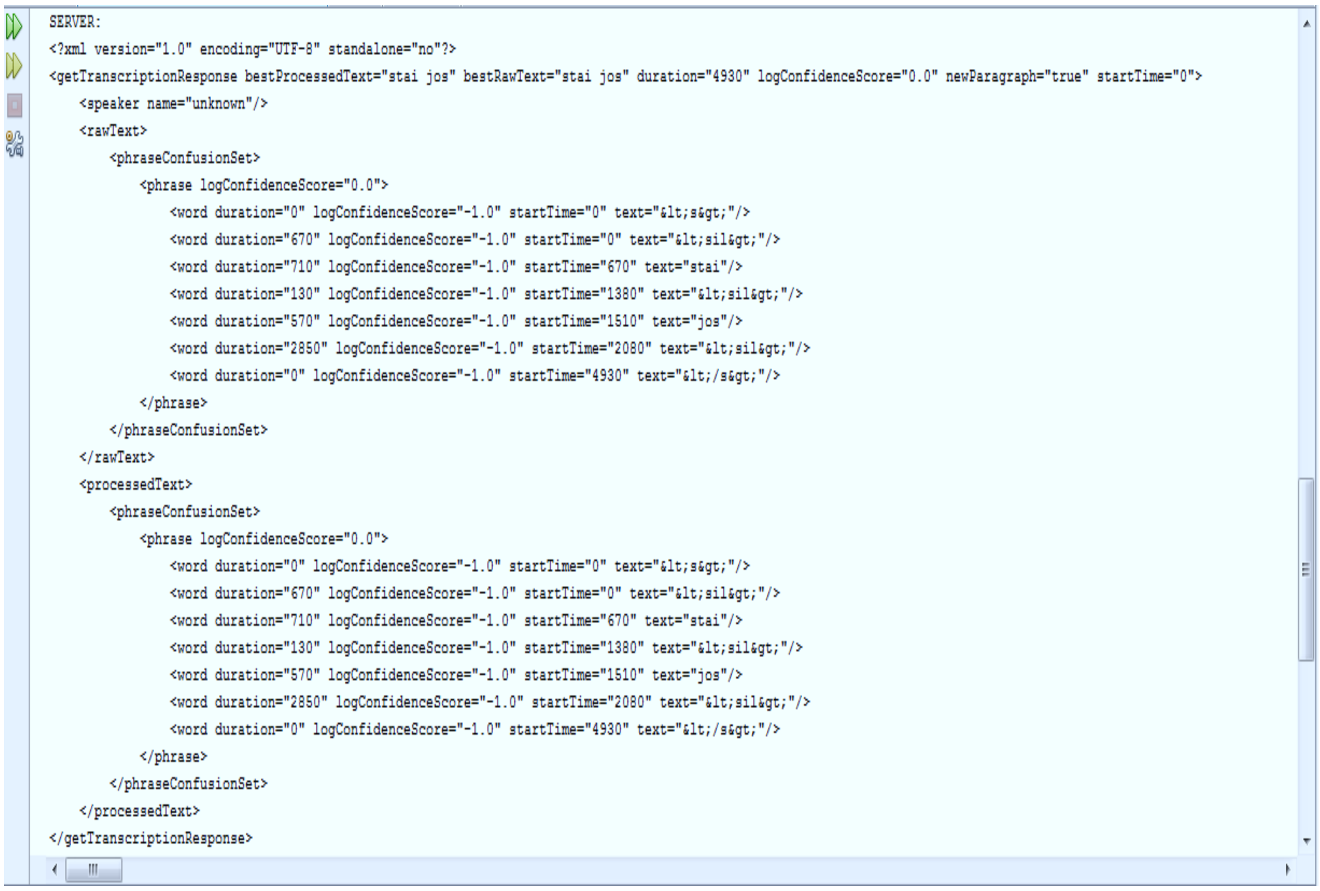
- ✓ The sender: Nao Aldebaran naoaldebaranrobot@gmail.com
- ✓ The subject: Demand
- ✓ The date: Sunday, 29 June, 15:23
- ✓ The content: No content, only the attachment, retrieved using the class javax.mail.internet.MimeMultipart



```
Message #1:
        From: Nao Aldebaran <naoaldebaranrobot@gmail.com>
        Subject: Demand
        Sent Date: Sun Jun 29 15:23:37 EEST 2014
        Message: javax.mail.internet.MimeMultipart@4f758781
```

Figure 4.10 Fetched e-mail details

The connection to the server is established, a series of messages are shown but the user is interested only in the response the server sends back to the client part. Figure 4.11 shows the response returned by the server, as an XML message.



```
SERVER:
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<getTranscriptionResponse bestProcessedText="stai jos" bestRawText="stai jos" duration="4930" logConfidenceScore="0.0" newParagraph="true" startTime="0">
    <speaker name="unknown"/>
    <rawText>
        <phraseConfusionSet>
            <phrase logConfidenceScore="0.0">
                <word duration="0" logConfidenceScore="-1.0" startTime="0" text="&lt;s&gt;"/>
                <word duration="670" logConfidenceScore="-1.0" startTime="0" text="&lt;sil&gt;"/>
                <word duration="710" logConfidenceScore="-1.0" startTime="670" text="stai"/>
                <word duration="130" logConfidenceScore="-1.0" startTime="1380" text="&lt;sil&gt;"/>
                <word duration="570" logConfidenceScore="-1.0" startTime="1510" text="jos"/>
                <word duration="2850" logConfidenceScore="-1.0" startTime="2080" text="&lt;sil&gt;"/>
                <word duration="0" logConfidenceScore="-1.0" startTime="4930" text="&lt;/s&gt;"/>
            </phrase>
        </phraseConfusionSet>
    </rawText>
    <processedText>
        <phraseConfusionSet>
            <phrase logConfidenceScore="0.0">
                <word duration="0" logConfidenceScore="-1.0" startTime="0" text="&lt;s&gt;"/>
                <word duration="670" logConfidenceScore="-1.0" startTime="0" text="&lt;sil&gt;"/>
                <word duration="710" logConfidenceScore="-1.0" startTime="670" text="stai"/>
                <word duration="130" logConfidenceScore="-1.0" startTime="1380" text="&lt;sil&gt;"/>
                <word duration="570" logConfidenceScore="-1.0" startTime="1510" text="jos"/>
                <word duration="2850" logConfidenceScore="-1.0" startTime="2080" text="&lt;sil&gt;"/>
                <word duration="0" logConfidenceScore="-1.0" startTime="4930" text="&lt;/s&gt;"/>
            </phrase>
        </phraseConfusionSet>
    </processedText>
</getTranscriptionResponse>
```

Figure 4.11 Response of the server

The last step is to send the robot an e-mail containing the transcription of the vocal command. When the e-mail was successfully sent, a message appears, „Email sent.”. When e-mail account will be further fetched by the robot, as its behavior implies and NAO will perform the command spoken earlier by the user.

In the same time, the MainApplication.java will be still running, and after sending the e-mail, the polling action will be resumed. No new e-mail will be found, unless the behavior in loaded again onto the robot and a new command in spoken to it, so the message displayed will be as figure 4.8 shows, printed at an time interval of one second, until the application is stopped.

CONCLUSIONS

The main objective of this thesis was to design and implement the architectural solution, as well as the necessary applications for NAO to respond to voice commands spoken in Romanian language. To achieve this goal, three main targets had to be considered: understand the construction principles of the existing ASR, as well as the communication protocol between any client and the server (on which ASR is implemented), determine the optimum language for the robot programming and implement NAO's behavior, and also developing the auxiliary application needed for connecting the robot and the server.

The solution proposed by the author of this paper was displayed in figure 3.1 (Chapter 3) and implied the development of two applications: **AutomationSystmSpeechRecognition.crg** (behavior loaded onto the robot) and the **MainApplication.java** (application running on the PC and connecting the robot and the server hosting the ASR).

This thesis presents the steps that were employed by the author of this paper in order to achieve the main goal. Chapter 1 presents a brief description of NAO's hardware and software main characteristics, as a theoretical support. The relation between the microcontrollers and the devices they command was highlighted, together with the software versatility of the robot and the programming languages it accepts. The author of this paper had to determine NAO's working principle in order to be able to develop the behavior.

Chapter 2 described the architecture of an automatic speech recognition system and its fundamental constitutive parts. As a starting point, the mechanism of speech formation was described, followed by the resources and instruments necessary building acoustic, phonetic and language models.

The contribution of the author is highlighted in Chapters 3 and 4, where the development of the proposed solution and its implementation are presented (as figure 3.1 exhibits). Chapter 3 focuses on the robot; the proposed solution is a NSR system (acquisition of the vocal signal in performed at Client level and decoding of the vocal signal in performed at Server level). NAO is in fact as the input and the output of the entire system. At its level, the acquisition of the vocal signal is done and the purchased file is send to a Java application; after the speech-to-text transcription, the transcript of the vocal command is received by the robot and the proper action is performed.

The author of this thesis established a simple commands list composed of five commands that mostly imply simple movements of NAO. The list was designed in such way that the words to be as different as possible one from another in order for the ASR to not introduce errors while transcription is performed. The project (NAO behavior), developed using Choregraphe software program is called **AutomationSystemSpeechRecognition.crg** and it was implemented in order to have several functionalities: record the vocal command spoken by the user, send an e-mail to asrserverspeed@gmail.com with the acoustic signal file attached, wait while the .wav file in converted into text, fetch the e-mail account naoaldebaranrobot@gmail.com for the response e-mail, perform the action indicated by the vocal command.

Chapter 4 is dedicated to the Client – Server application. The author contributions consist in implementing a Java application (**MainApplication.java**), using a set of protocols, with the following flow: access the e-mail server, fetch the e-mail message that was previously send by the robot and save the attachment (representing in fact the vocal command), send data to the server (using the existing protocol), receive the transcription of the vocal command (transmitted data) and send via e-mail plain text message the speech-to-text sequence back to the robot.

In order to implement the last application, two important elements have been used: a series of protocols (for e-mail sending and reading – POP3 and SMTP; for communication with the server – implemented with XML messages) and JavaMail, a platform-independent and protocol-independent framework to build mail and messaging applications, providing facilities to send and read e-mails.

The personal contribution of the author can be summarized as follows:

- ✓ Creating an achievable architectural solution for the main objective;
- ✓ Establishing the commands list for NAO, given the particularities and the performances of the ASR;
- ✓ Deciding the proper solution to be implemented on the robot – a timeline behavior;
- ✓ Designing and creating the Choregraphe behavior (AutomationSystemSpeechRecognition.crg), application running on NAO and making it perform the following actions: record the vocal command, send it through e-mail (as an attachment) to a certain recipient, wait while the speech-to-text transcription is performed, fetch the e-mail for the command sent as plain text, perform the indicated action;

- ✓ Deciding the use of a reliable programming language for the application development;
- ✓ Designing and creating a Java application (MainApplicatin.java) with the following capabilities: access the e-mail server (corresponding to the recipient account), fetch the e-mail message that was previously send by the robot and save the attachment (representing in fact the vocal command), send data to the server (using the existing protocol), receive the transcription of the vocal command (transmitted data) and send via e-mail plain text message the speech-to-text sequence back to the robot;
- ✓ Adapting the existing communication protocol to the server and the existing application to the MainApplication.java.

The solution proposed by the author of this paper is a practical and reliable solution, but it was designed as a proof of concept, further development being required. The testing phase was complex and performed continuously, in order to improve the entire architecture. Both applications reach their goal 100% of the time; but during some tests the robot was not able to perform the spoken command because of the incorrect transcription performed by the server. Further improvements are needed for the ASR mechanism.

The automatic speech recognition domain is still a trending research domain and moreover, its applicability became necessary in the era we live. Development of a proficiency ASR for Romanian language is a must, in order to be integrated in devices and to be used at a national scale.

NAO represents an excellent research tool; unfortunately, the robot acquisition was done in February 2014, so the available period to understand its construction principle, the link between the hardware and the software parts, as well as the programming tools and available programming languages was too short. A major drawback in working with such an innovative and new tool was the lack of documentation; besides the official documentation provided by Aldebaran Robotics, fewer articles were found in the literature, because it represents a pretty new field of research, accessible to a limited number of researchers.

Further improvements can be developed on different directions:

- ✓ The development of a more complex behavior, allowing NAO to continuously record and then send the .wav file to the server, implying a bigger data flux,
- ✓ The recognition should be independent on the speaker such that any person could command NAO; this means a more general acoustic model must be created (obtained be gathering resources from a non-homogenous group of people).
- ✓ The connection between NAO and the server on which ASR is running should be made directly, through sockets.
- ✓ The implementation of an ASR directly on NAO; it implies the development of such an architecture considering NAO’s resources, but eases the architectural solution of the entire system, by eliminating the client-server architecture.
- ✓ The integration of NAO inside the operating system of a smart house; the response to the vocal commands will be actions linked to the house feature (open/close the light, raise the temperature, etc.).

REFERENCES

[1] http://en.wikipedia.org/wiki/Humanoid_robot - accessed on 13.06.2014

[2] Pfiefer, R., Scheier, C., “Understanding Intelligence”, Bradford Books, 2011

[3] http://www.aldebaran.com/ -accessed on 14.06.2014

[4] Gouaillier, D., Hugel, V., Blazevic, P., Kilner,C., Monceaux, J., Lafourcade, P., Marnier, B., Serre, J., Maisonnier, B., *“Mechatronic design of NAO humanoid”,* 2009 IEEE International Conference on Robotics and Automation, Kobe, Japan, 2009

[5] NAO Datasheet

[6] https://www-ssl.intel.com/content/www/us/en/processors/atom/atom-z540-z530-z520-z510-z500-45-nm-technology-datasheet.html - accessed on 16.06.2014

[7] Sanchez, T., *“Artificial vision in the NAO humanoid robot*”, Ph. D. Thesis, University Rovira I Virgili, 2009

[8] Cucu, H., *“Towards a speaker-independent, large-vocabulary continuous speech recognition system for Romanian”*, Ph.D. Thesis, University Politehnica of Bucharest, 2011

[9] Stuckless, R., *“Developments in real-time speech-to-text communication for people with impaired hearing”*, In M. Ross (Ed.), *“Communication access for people with hearing loss”*, pp. 197-226, MD: York Press, Baltimore, 1994

[10] Baker, J., *“The DRAGON system – an overview”*, IEEE Transactions on Acoustics, Speech and Signal Processing, pp. 24-29, vol. 23, no. 1, 1975

[11] Huang, X., Acero, A., Hon, H.-W., *“Spoken Language Processing Guide to Theory, Algorithm, and System Development”*, Prentice Hall, 2001

[12] Flanagan, J.L., *“Speech Analysis Synthesis and Perception”*, Springer-Verlag Berlin Heidelberg, 1972

[13] http://en.wikipedia.org/wiki/Speech_production#mediaviewer/File:Illu01_head_neck.jpg – accessed on 11.06.2014

[14] O'Shaughnessy, D., “Linear Predictive Coding”, IEEE Potentials, pp. 29-32, vol. 7, 1988

[15] Hermansky, H., *“Perceptual Linear Predictive (PLP) Analysis of Speech”*, Journal of the Acoustical Society of America, vol. 87, no. 4, pp. 1738-1752, 1990

[16] Davis, S.B., Mermelstein, P., *"Comparison of Parametric Representations for Monosyllabic Word Recognition in Continuously Spoken Sentences"* in IEEE Transactions on Acoustics, Speech, and Signal Processing, 28(4), pp. 357–366, 1980

[17] Reynolds, D.A., Rose, R.C., *“Robust Text-Independent Speaker Identification using Gausian Mixture Models”*, IEEE Transactions on Acoustic, Speech and Signal processing, vol. 3, pp. 72-83, 1995

[18]https://community.aldebaran.com/doc/1-14/software/choregraphe/choregraphe_overview.html – accessed on 01.07.2014

[19] NAO – Documentation & Software

[20] http://en.wikipedia.org/wiki/Client%E2%80%93server_model – accessed on 26.06.2014

[21] http://www.oracle.com/technetwork/java/javamail/index.html – accessed on 29.06.2014

[22] https://javamail.java.net/nonav/docs/api/ – accessed on 29.06.2014

[23] Oracle America, Inc., *“JavaMail API Design Specification”*, version 1.5, 2013

APPENDIX 1

```
[y, Fs] = wavread('Proposition.wav');

t=0:1/Fs:length(y)/Fs-1/Fs;
figure(1)
plot(t,y); xlabel('T[s]'); ylabel('A');


figure(2)
spectrogram(y(:,1),256,250,256,Fs,'yaxis');
```

APPENDIX 2

```java
package org.etti.speech.transcriber.client;

import java.io.File;
import java.io.IOException;

import java.security.NoSuchAlgorithmException;

import java.util.Properties;
import java.util.logging.Level;
import java.util.logging.Logger;
import java.util.Date;

import javax.mail.Authenticator;
import javax.mail.Address;
import javax.mail.Folder;
import javax.mail.Message;
import javax.mail.MessagingException;
import javax.mail.Multipart;
import javax.mail.NoSuchProviderException;
import javax.mail.PasswordAuthentication;
import javax.mail.Part;
import javax.mail.Session;
import javax.mail.Store;
import javax.mail.Transport;
import javax.mail.internet.AddressException;
import javax.mail.internet.InternetAddress;
import javax.mail.internet.MimeMessage;
import javax.mail.internet.MimeBodyPart;
import javax.xml.parsers.ParserConfigurationException;
import javax.xml.transform.TransformerException;

import org.xml.sax.SAXException;

/**

 *
 * @author Diana Sandru
 *
 */
public class MainApplication {
    private String saveDirectory;
    private String liveTranscriberServerAddress;
    private int liveTranscriberServerPort;

    private static TranscriberClient _client;

    /**
     * Sets the directory where attached files will be stored.
     * @param dir absolute path of the directory
     */
    public void setSaveDirectory(String dir) {
        this.saveDirectory = dir;
    }

    /**
     * mainLoop contains the loop that performs the pooling action and it is
     * composed of: download new e-mails & save attachment, send the attachment
to the server and
     * the send the received transcription via e-mail
     *
     * These parameters are used to configure the e-mail address
     * containing the e-mails received from the robot
     * @param host
     * @param port
     * @param userName
     * @param password
     *
     */
    public void mainLoop(String host, String port,
            String userName, String password) throws InterruptedException {
        Properties properties = new Properties();

        // Set POP3 server properties
        properties.put("mail.pop3.host", host);
        properties.put("mail.pop3.port", port);
        properties.put("mail.pop3disabletop", "true");

        // SSL settings
        properties.setProperty("mail.pop3.socketFactory.class",
                "javax.net.ssl.SSLSocketFactory");
        properties.setProperty("mail.pop3.socketFactory.fallback", "false");
        properties.setProperty("mail.pop3.socketFactory.port",
                String.valueOf(port));

        Session session = Session.getDefaultInstance(properties, null);

        System.out.println("Ready...");

        while (true) {
            try {
            // Connect to the message store
            Store store = session.getStore("pop3");
            store.connect(host, userName, password);

            System.out.println("Email checking...");

            // Open the inbox folder
            Folder folderInbox = store.getFolder("INBOX");
            folderInbox.open(Folder.READ_ONLY);

            // Fetch new messages from server
            Message[] arrayMessages = folderInbox.getMessages();

            System.out.println("New emails: " + arrayMessages.length);
            for (int i = 0; i < arrayMessages.length; i++) {
                Message message = arrayMessages[i];
                Address[] fromAddress = message.getFrom();
                String from = fromAddress[0].toString();
                String subject = message.getSubject();
                String sentDate = message.getSentDate().toString();

                String contentType = message.getContentType();
                String messageContent = "";

                // Store attachment file name, separated by comma
                String attachFiles = "";

                if (contentType.contains("multipart")) {
                    // if content contains attachments
                    Multipart multiPart = (Multipart) message.getContent();
                    int numberOfParts = multiPart.getCount();
                    for (int partCount = 0; partCount < numberOfParts;
partCount++) {
                        MimeBodyPart part = (MimeBodyPart)
multiPart.getBodyPart(partCount);
                        if
(Part.ATTACHMENT.equalsIgnoreCase(part.getDisposition())) {

                            // Extract the attachment
                            String fileName = part.getFileName();
                            part.saveFile(saveDirectory + File.separator +
fileName);

                            String mailMessage = " ";

                            // Call the server with the extracted attachment
                            try {
                                mailMessage =
_client.testCaseScenario1(14,saveDirectory + File.separator + fileName);
                            } catch (ParserConfigurationException ex) {

Logger.getLogger(MainApplication.class.getName()).log(Level.SEVERE, null, ex);
                            } catch (TransformerException ex) {

Logger.getLogger(MainApplication.class.getName()).log(Level.SEVERE, null, ex);
                            } catch (SAXException ex) {

Logger.getLogger(MainApplication.class.getName()).log(Level.SEVERE, null, ex);
                            } catch (NoSuchAlgorithmException ex) {

Logger.getLogger(MainApplication.class.getName()).log(Level.SEVERE, null, ex);
                            }

                            // Set the outgoing e-mail message information
                            String host2 = "smtp.gmail.com";
                            String port2 = "587";
                            String mailFrom = "asrserverspeed@gmail.com";
                            String password2 = "diplomathesis2014";
                            String mailTo = "naoaldebaranrobot@gmail.com";
                            String subject2 = "Response";

                            // Send the outgoing e-mail message - back to the
robot

                            try {
                                sendEmail(host2, port2, mailFrom, password2,
mailTo,
                                        subject2, mailMessage);
                                System.out.println("Email sent.");
                            } catch (Exception ex) {
                                System.out.println("Failed to sent email.");
                                ex.printStackTrace();
                            }

                        } else {
                            messageContent = part.getContent().toString();
                        }
                    }
                }

                // Print out details of each fetched e-mail message
                System.out.println("Message #" + (i + 1) + ":");
                System.out.println("\t From: " + from);
                System.out.println("\t Subject: " + subject);
                System.out.println("\t Sent Date: " + sentDate);
                System.out.println("\t Message: " + messageContent);

            }

            // Disconnect
            folderInbox.close(false);
            store.close();
        } catch (NoSuchProviderException ex) {
            System.out.println("No provider for pop3.");
            ex.printStackTrace();
        } catch (MessagingException ex) {
            System.out.println("Could not connect to the message store");
            ex.printStackTrace();
        } catch (IOException ex) {
            ex.printStackTrace();
        }
            // Pooling action done at intervals of 1 second
            Thread.sleep(1000);
        }
    }

    /**
     * Function to send the response by e-mail to the robot
     *
     * @param host
     * @param port
     * @param userName
     * @param password
     * @param toAddress
     * @param subject
     * @param message
     * @throws AddressException
     * @throws MessagingException
     */
    public void sendEmail(String host, String port,
            final String userName, final String password, String toAddress,
            String subject, String message) throws AddressException,
            MessagingException {

        // Set SMTP server properties
        Properties properties = new Properties();
        properties.put("mail.smtp.host", host);
        properties.put("mail.smtp.port", port);
        properties.put("mail.smtp.auth", "true");
        properties.put("mail.smtp.starttls.enable", "true");
```

```
        // Create a new session with an authenticator
        Authenticator auth = new Authenticator() {
            public PasswordAuthentication getPasswordAuthentication() {
                return new PasswordAuthentication(userName, password);
            }
        };

        Session session = Session.getInstance(properties, auth);

        // Create a new e-mail message
        Message msg = new MimeMessage(session);

        msg.setFrom(new InternetAddress(userName));
        InternetAddress[] toAddresses = { new InternetAddress(toAddress) };
        msg.setRecipients(Message.RecipientType.TO, toAddresses);
        msg.setSubject(subject);
        msg.setSentDate(new Date());
        // Set plain text message
        msg.setText(message);

        // Sends the e-mail
        Transport.send(msg);

    }


    public static void main(String[] args) throws InterruptedException {

        // Set the ingoing e-mail message information
        // For the account on which the pooling action will be performed


        String liveTranscriberServerAddress = "141.85.252.230";
        int liveTranscriberServerPort = 5004;

        _client = new TranscriberClient(liveTranscriberServerAddress,
liveTranscriberServerPort);

        System.out.println("Start...");

        String host = "pop.gmail.com";
        String port = "995";
        String userName = "asrserverspeed@gmail.com";
        String password = "diplomathesis2014";

        String saveDirectory = "S:/Thesis/Attachments";

        MainApplication receiver = new MainApplication();
        receiver.setSaveDirectory(saveDirectory);
        receiver.mainLoop(host, port, userName, password);

    }
}
```

APPENDIX 3

```java
package org.etti.speech.transcriber.client;

import java.io.*;
import java.net.Socket;
import java.nio.file.*;
import java.security.NoSuchAlgorithmException;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.DOMSource;
import javax.xml.transform.stream.StreamResult;
import org.etti.speech.io.*;
import org.etti.speech.transcriber.struct.*;
import org.etti.speech.transcriber.util.XMLBuilder;
import org.etti.speech.transcriber.util.XMLParser;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.xml.sax.SAXException;

/**
 * //http://www.mkyong.com/java/how-to-read-xml-file-in-java-dom-parser/
 * //http://www.mkyong.com/java/how-to-create-xml-file-in-java-dom/
 *
 *
 * @author cucu
 */
public class TranscriberClient {
    public static final String FILE = "S:/Thesis/Attachments/Record.wav";
    public static final int DOMAIN = 14;

    private final String liveTranscriberServerAddress;
    private final int liveTranscriberServerPort;

    public TranscriberClient(String _liveTranscriberServerAddress, int
_liveTranscriberServerPort){
        liveTranscriberServerAddress = _liveTranscriberServerAddress;
        liveTranscriberServerPort = _liveTranscriberServerPort;
    }

    public static void main(String _args[]) throws Exception {
        String _liveTranscriberServerAddress = "141.85.252.230";
        int _liveTranscriberServerPort = 5004;
        TranscriberClient _client = new
TranscriberClient(_liveTranscriberServerAddress, _liveTranscriberServerPort);

       String _response = _client.testCaseScenario1(DOMAIN, FILE);
    }

//--------------------------------------------------------------------------
-------
    public String testCaseScenario1(int _asrDomainId, final String
_audioFileName) throws IOException, InterruptedException,
ParserConfigurationException, TransformerConfigurationException,
TransformerException, SAXException, NoSuchAlgorithmException {
        Socket _socket = new Socket(liveTranscriberServerAddress,
liveTranscriberServerPort);
        XMLOutputStream _outputStream = new
XMLOutputStream(_socket.getOutputStream());
        XMLInputStream _inputStream = new
XMLInputStream(_socket.getInputStream());
        System.out.println("Successfully connected to LiveTranscriberServer");

        DocumentBuilder _documentBuilder =
DocumentBuilderFactory.newInstance().newDocumentBuilder();
        Transformer _transformer =
TransformerFactory.newInstance().newTransformer();
        _transformer.setOutputProperty(OutputKeys.INDENT, "yes");
        _transformer.setOutputProperty("{http://xml.apache.org/xslt}indent-
amount", "4");
        Document _requestDocument;

    //Send an autheticateRequest
        _requestDocument = XMLBuilder.createAuthenticateRequest("diana.sandru",
"mewn$#11p");
        _transformer.transform(new DOMSource(_requestDocument), new
StreamResult(_outputStream));
        _outputStream.send();
        System.out.println("\n//-------------------------------------\nCLIENT:");
        _transformer.transform(new DOMSource(_requestDocument), new
StreamResult(System.out));

    //Receive the authenticateResponse xml and print it on the screen
        _inputStream.receive();
        System.out.println("\nSERVER:");
        _transformer.transform(new
DOMSource(_documentBuilder.parse(_inputStream)), new StreamResult(System.out));

    //Send a getSupportedConfigurationsRequest
        _requestDocument = XMLBuilder.createGetSupportedConfigurationsRequest();
        _transformer.transform(new DOMSource(_requestDocument), new
StreamResult(_outputStream));
        _outputStream.send();
        System.out.println("\n//-------------------------------------\nCLIENT:");
        _transformer.transform(new DOMSource(_requestDocument), new
StreamResult(System.out));

    //Receive the getSupportedConfigurationsResponse xml and print it on the
screen
        _inputStream.receive();
        System.out.println("\nSERVER:");
        _transformer.transform(new
DOMSource(_documentBuilder.parse(_inputStream)), new StreamResult(System.out));

    //Send a getAudioDataPortRequest
```

```
        _requestDocument = XMLBuilder.createGetAudioDataPortRequest();
        _transformer.transform(new DOMSource(_requestDocument), new
StreamResult(_outputStream));
        _outputStream.send();
        System.out.println("\n//------------------------------------\nCLIENT:");
        _transformer.transform(new DOMSource(_requestDocument), new
StreamResult(System.out));

    //Receive the getAudioDataResponse XML, get the port and print XML document
on the screen
        _inputStream.receive();
        Document _responseDocument = _documentBuilder.parse(_inputStream);
        Element _responseElement = _responseDocument.getDocumentElement();
        int _audioDataPort =
Integer.parseInt(_responseElement.getAttribute(ProtocolConfig.ATTRIBUTE_PORT));
        System.out.println("\nSERVER:");
        _transformer.transform(new DOMSource(_responseDocument), new
StreamResult(System.out));

    //Connect to the audioDataSocket
        Socket _audioDataSocket = new Socket(liveTranscriberServerAddress,
_audioDataPort);
        final OutputStream _audioDataOutputStream =
_audioDataSocket.getOutputStream();

    //Send a getTranscriptionRequest
        _requestDocument =
XMLBuilder.createGetTranscriptionRequest(_asrDomainId, "PCM_SIGNED", "wide", new
TranscriptionOptions(true, true, true, true, true));
        _transformer.transform(new DOMSource(_requestDocument), new
StreamResult(_outputStream));
        _outputStream.send();
        System.out.println("\n//------------------------------------\nCLIENT:");
        _transformer.transform(new DOMSource(_requestDocument), new
StreamResult(System.out));

    //Receive a startTranscriptionAck. Print it on the screen
        _inputStream.receive();
        System.out.println("\nSERVER:");
        _transformer.transform(new
DOMSource(_documentBuilder.parse(_inputStream)), new StreamResult(System.out));

    //Start sending audio data. Asynchronously!
        new Thread(){
            @Override
            public void run(){
                Path _audioFile = Paths.get(_audioFileName);

                try (InputStream _inputStream =
Files.newInputStream(_audioFile)){
                    byte[] _buffer = new byte[8192];
                    int _length, _totalLength = 0;
                    while ((_length = _inputStream.read(_buffer)) != -1) {
                        _audioDataOutputStream.write(_buffer, 0, _length);
                    }
                    _audioDataOutputStream.close();
                } catch (IOException _exc) {
                    _exc.printStackTrace();
                }
            }
        }.start();

        String _response = null;
    //Receive several getTranscriptionResponses. Print them on the screen
        boolean _receivedDoneTranscriptionAck = false;
        while (!_receivedDoneTranscriptionAck){
            _inputStream.receive();
            _responseDocument = _documentBuilder.parse(_inputStream);
            _responseElement = _responseDocument.getDocumentElement();
            _receivedDoneTranscriptionAck =
_responseElement.getNodeName().equals(ProtocolConfig.ACK_DONE_TRANSCRIPTION);
            System.out.println("\nSERVER:");
            _transformer.transform(new DOMSource(_responseDocument), new
StreamResult(System.out));
            if (!_receivedDoneTranscriptionAck){
                Transcription _transcription =
XMLParser.getTranscription(_responseElement);
                _response = _transcription.getBestRawText();
            }
        }

    //Send a correctedTranscriptionRequest
        _requestDocument = XMLBuilder.createSetCorrectedTranscriptionRequest(1,
"correctedRawText", "correctedProcessedText");
        _transformer.transform(new DOMSource(_requestDocument), new
StreamResult(_outputStream));
        _outputStream.send();
        System.out.println("\n//------------------------------------\nCLIENT:");
        _transformer.transform(new DOMSource(_requestDocument), new
StreamResult(System.out));

    //Receive a setCorrectedTranscriptionAck. Print it on the screen
        _inputStream.receive();
        System.out.println("\nSERVER:");
        _transformer.transform(new
DOMSource(_documentBuilder.parse(_inputStream)), new StreamResult(System.out));

        return _response;
    }
}
```