Universitatea Politehnica București
Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Sistem Complet de Automatizare

Lucrare de Licență

prezentată ca cerință partială pentru obținerea *Titlului de Inginer* în domeniul *Electronică, Telecomunicații și Tehnologia Informației*, programul de studii *Rețele și Software pentru Telecomunicații*

Conducător științific
Prof. Corneliu BURILEANU

Absolvent
Ana-Maria RADU

Anexa 1

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Departamentul Telecomunicaţii

Aprobat Director de Departament :
Prof. Dr. Ing. Silviu Ciochină

TEMA PROIECTULUI DE DIPLOMĂ
a studentului Radu T. L. Ana- Maria, grupa 441 D

1. Titlul temei: *Sistem complet de automatizare*

2. Contribuţia practică, originală a studentului va consta în:
Acest proiect de diplomă îşi propune realizarea conceptuală şi implementarea unui sistem complet de automatizare, structurat în 3 blocuri principale:

1) *Sistem de achiziţie şi prelucare - constituit din senzori, periferice aferente şi modulul de prelucrarea semnalului capturat*
2) *Sistem de comandă şi control - va primi la intrare mesaje sub o formă bine determintă, le va interpreta şi prelucra pentru a trimite comenzile adecvate dispozitivului terminal*
3) *Element de execuţie - terminalul ce va executa comenzile emise la pasul anterior*

Proiectul va beneficia şi de realizare practică, luându-se în considerare atât componentele specifice fiecărui bloc, din punct de vedere hardware, interconectarea/ comunicaţia dintre acestea, dar şi programele aferente.

3. Proiectul se bazează pe cunoştinţe dobândite în principal la urmatoarele 3-4 discipline: *Microcontrolere, Arhitectura Microprocesoarelor, Prelucrarea Digitala a Semnalelor, Tehnologii de Programare în Internet*

4. Proprietatea intelectuală asupra proiectului aparţine: *Studentului, Conducatorului de lucrare*

5. Realizarea practică/ proiectul ramân în proprietatea: *Studentului, UPB*

6. Locul de desfăşurare a activităţii: *UPB, Sala 3 de Calculatoare*

7. Data eliberării temei: *1. 10. 2014*

CONDUCATOR LUCRARE: **STUDENT:**

Prof. Corneliu BURILEANU **Ana- Maria RADU**

Anexa 5

Declaraţie de onestitate academică

Prin prezenta declar că lucrarea cu titlul "*Sistem complet de automatizare*", prezentată în cadrul Facultăţii de Electronică, Telecomunicaţii şi Tehnologia Informaţiei a Universităţii "Politehnica" din Bucureşti ca cerinţă parţială pentru obţinerea titlului de *Inginer* în domeniul *Electronică, Telecomunicații și Tehnologia Informației*, programul de studii *Rețele și Software pentru Telecomunicații* este scrisă de mine şi nu a mai fost prezentată niciodată la o facultate sau instituţie de învăţământ superior din ţară sau străinătate.

Declar că toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referinţe bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar şi în traducere proprie din altă limbă, sunt scrise între ghilimele şi fac referinţă la sursă. Reformularea în cuvinte proprii a textelor scrise de către alţi autori face referinţă la sursă. Înţeleg că plagiatul constituie infracţiune şi se sancţionează conform legilor în vigoare.

Declar că toate rezultatele simulărilor, experimentelor şi măsurătorilor pe care le prezint ca fiind făcute de mine, precum şi metodele prin care au fost obţinute, sunt reale şi provin din respectivele simulări, experimente şi măsurători. Înţeleg că falsificarea datelor şi rezultatelor constituie fraudă şi se sancţionează conform regulamentelor în vigoare.

Bucureşti, *03. 07. 2015*

Absolvent *Ana-Maria RADU*

_____

(semnătura în original)

Cuprins

LISTA DE FIGURI

LISTA DE TABELE

LISTA DE ACRONIME

A
ABS – Acrilonitril Butadien Stiren
API – Application Programming Interface

D
DCM – Device Communication Manager

H
HMM – Hidden Markov Models

I
IP – Internet Protocol

M
MFCCs – Mel-Frequency Cepstrum Coefficients

O
OOV – Out of vocabulary
OS – Operating System

P
P2P – Peer-to-peer (Punct la punct)
PC – Personal Computer
PCM – Pulse Code Modulation

R
RAM – Random Access Memory
RAV – Recunoaștere Automată a Vorbirii
RGB – Red-Green-Blue (Roșu, Verde, Albastru)

S
S2T – Speech-to-Text
SER – Sentence error rate
SHA – Secure Hash Algorithm
SSH – Secure Shell

T
TCP – Transmission Control Protocol

U
UCP – Unitate Centrală de Prelucrare
UDP – User Datagram Protocol

W
WAV – Waveform Audio File Format
WER – Word error rate

X
XML – Extensible Markup Language

INTRODUCERE

Încă din era preistorică până în zilele noastre comunicarea a fost, este și va fi principala caracteristică a omenirii. Interdependența om - comunicare stă la baza societății, fără aceasta, existența omului, ființă ce interacționează și relaționează în mod dominant prin intermediul vorbirii, ar fi fost compromisă.

La început, oamenii puteau comunica doar pe distanțe scurte, adresându-se reciproc unul altuia. De-a lungul timpului, vorbirea umană a stârnit interesul oamenilor de știință, atât în domeniul medical, dar mai ales în domeniul ingineriei. Odată cu progresul tehnologiei, inginerii au dezvoltat metode de sintetizare a vorbirii, cât și modalități de a o transporta pe distanțe extraordinar de mari.

În această lume plină de contradicții și contraste, probabil cea mai remarcabilă evoluție o au tehnologiile de comunicații și informații, evoluție complexă, atat hardware, cât și software. Nu e de conceput mediu de activitate umană care să nu depindă de acestea, întreaga infrastructură de comunicație și prelucrarea informațiilor tinzând spre globalizarea absolută. În ultimul secol, calculatoarele au demonstrat o ascensiune uimitoare, ajungând la puteri de calcul foarte mari și la numeroase funcționalități menite să faciliteze munca omului modern. Însă, pentru a le putea utiliza optim au trebuit dezvoltate o multitudine de interfețe care să ușurze legatura om - mașină.

De asemenea, aflându-ne în „era vitezei”, lipsa timpului a devenit o problemă majoră, omul tinzând spre tehnologii care îi pot permite desfașurarea unor activități fără prea multe eforturi fizice, cât mai rapid. Drept urmare, a luat naștere conceptul de comandă vocală. Această nouă idee a fost acceptată cu foarte mare entuziasm, ajungând într-un ritm alert, de la stadiul de prototip la cel de caracteristică implementabilă. De exemplu, aproape toate dispozitivele apărute în ultimul deceniu suportă acestă particularitate, dispozitive precum telefoane inteligente, tablete, roboți etc.; toate sunt capabile să răspundă comenzilor vocale. Mai mult decât atât, ideea de comandă vocală reprezintă unul din fundamentele noțiunii de „casă inteligentă”, unde idealul tinde a fi acela ca toate comenzile date de om sa fie prin voce.

Omul din ziua de astăzi are o afinitate și pune mare preț pe tot ce înseamnă inovație, așadar, interacțiunea om - robot nu mai aparține de domeniul fantasticului. Dezvoltarea roboților umanoizi, care imită formele și funcțiile corpului uman, fac această interacțiune și mai interesantă. Un exemplu de asemenea robot este oferit de firma Aldebaran Robotics, robotul NAO. Acesta este unul extrem de complex, îndeplinind toate caracteristicile unui robot umanoid. Lansat pentru prima dată in anul 2006 și evoluat de la o generație la alta, NAO se dorește a fi „prietenul tuturor”, așa cum susțin cei de la Aldebaran Robotics [1].

Revenind la domeniul recunoașterii vocale, ramura aceasta este una extrem de cercetată și dezvoltată la nivel mondial, cât și național. Particularitățile oferite de fonetica fiecărei limbi au condus la dezvoltarea sistemelor de Recunoaștere Automată a Vorbirii (RAV), în mod diferențiat. Drept urmare, autenticitatea limbii române a fost și este intens studiată de echipa laboratorului *Speech and Dialogue* (SpeeD), un asemenea sistem fiind implementat deja, în premieră pentru limba noastră.

Motivația principală în alegerea acestei teme de licență, *Sistem Complet de Automatizare*, este bazată pe interesul personal către cele două arii prezentate anterior, robotul NAO și sistemul de Recunoaștere Automată a Vorbirii, dar mai ales pe provocarea de a le combina în vederea obținerii unui rezultat compact.

Scopul principal al acestei lucrări este de a îngloba cele două concepte într-un Sistem Complet Automat, astfel încât robotul NAO să fie capabil de a răspunde unor comenzi predefinite în limba română. Cercetând cele implementate deja în această direcție, un scop secundar al tezei este acela de a îmbunătăți arhitectura de comunicație între robotul NAO și serverul pe care rulează serviciul de Recunoaștere Automată a Vorbirii.

În vederea dezvoltării acestui sistem s-a urmărit schema bloc funcțională:

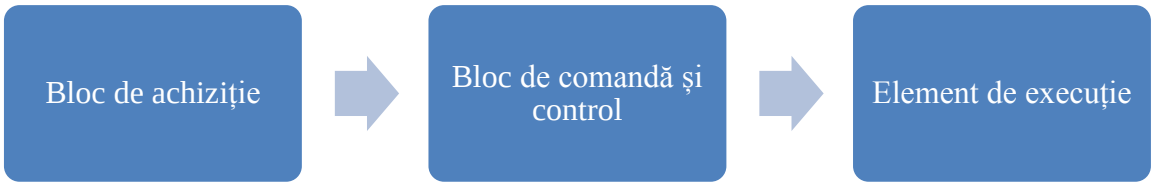


Figura 0.1 Schema bloc funcțională a Sistemului Complet de Automatizare

Pentru realizarea acestui sistem se vor considera următoarele obiective punctuale:

- înțelegerea principiilor conceptuale ale sistemului de recunoaștere automată a vorbirii și a protocolului impus de aplicația de transcriere audio-text;
- determinarea soluției optime de implementare și programare a comportamentului robotului NAO.

Această lucrare scrisă este structurată în patru capitole principale. Primul capitol prezintă caracteristicile, atât hardware cât și software, ale robotului NAO. Cel de-al doilea, descrie aspectele teoretice ale sistemului de Recunoaștere Automată a Vorbirii. Ultimele două surprind contribuția autorului la dezvoltarea practică a Sistemului Complet de Automatizare.

Contribuția personală a autorului poate fi rezumată astfel:

- Conceperea structurii Sistemului Complet de Automatizare;
- Stabilirea și implementarea arhitecturii optime de comunicație robot – server, conform protocolului folosit de aplicația de recunoaștere automată a vorbirii;
- Elaborarea listei predefinite de comenzi, utilizată pentru exemplificarea practică;
- Prelucrarea răspunsului primit de la aplicația de recunoaștere automată a vorbirii;
- Stabilirea comportamentului robotului conform comenzilor recepționate – dezvoltarea programului software necesar controlului.

CAPITOLUL 1

NAO – ROBOT UMANOID AUTONOM

Un robot *umanoid* se identifică, în primul rând, prin forma sa similară cu cea a corpului uman. Se pot enumera o multitudine de scopuri ce pot viza acest design particular, însă cele mai importante sunt cele funcționale, cum ar fi interacțiunea cu diverse obiecte de uz general pentru oameni, dar și cele experimentale, ca de exemplu studierea locomoției bipede. În general, roboții umanoizi sunt constituiți din cap, torace, două mâini, două picioare. Totuși, unele versiuni pot modela doar o parte a corpului uman, de exemplu bustul și capul. De asemenea, unele versiuni de roboți pot fi specializați în anumite particularități ale corpului uman, cum ar fi existența unui cap complex elaborat cu scopul de a replica caracteristicile faciale, gură sau ochi. Cercetările din ultimii ani se axeaza tot mai mult pe dezvoltarea roboților androizi, roboți umanoizi construiți estetic pentru a reproduce cât mai detaliat atat corpul, cât și acțiunile unui om.[2]

Termenul *autonom*, în ceea ce privește roboții, se referă la capabilitățile acestora de a îndeplini funcții sub un anumit grad de autonomie. Autonomie înseamnă abilitatea de a se administra, mai exact independența controlului. Această caracteristică implică faptul că termenul definit mai sus este o propietate a unei relații între doi agenți, în cazul de față, între designer și robotul autonom. Independența, învățarea și dezvoltarea sunt factori ce duc la ridicarea gradului de autonomie a unui agent. [3]

Luând în considerare toate cele menționate anterior, se poate afirma cu certitudine că robotul NAO este unul *umanoid*, *autonom* și *biped*, de dimensiuni reduse, constituit din cap, cutie toracică, două mâini și două picioare. În cele ce urmează, se vor descrie pe larg toate caracteristicile acestuia.

1.1 CARACTERISTICI GENERALE

Conform celor relatate de echipa firmei Aldebaran Robotics, NAO este un robot conceput cu scopul de a deveni o companie agreabilă și prietenoasă în orice casă. El este capabil să meargă pe distanțe rezonabile, să recunoască persoanele, atât facial, cât și după voce, să asculte și chiar să vorbească. De la prima lansare, în anul 2006, a evoluat în mod constant, de la o generație la alta, cu scopul principal de a fi pe placul utilizatorilor, câștigandu-le prietenia, devenind un adevărat

camarad. Ca pas următor, în viitorul apropiat, se dorește integrarea robotului NAO în conceptul de *casă inteligentă*, concept modern aflat în plină evoluție.

Figura 1.2.1 Robotul NAO [1]

După cum se poate vizualiza în figura 1.1.1, robotul NAO este unic datorită design-ului și formei sale umanoide, dar și mobilității complexe, evidențiate de multitudinea încheieturilor prezente la toate membrele sale.

De asemenea, unicitatea sa este susținută și de faptul că în mai bine de 70 de țări NAO este utilizat ca suport didactic pentru cursurile de știința calculatoarelor și robotică din școlile primare până în universități. Cu ajutorul lui, elevii și studenții pot învăța și aprofunda limbaje de programare într-o manieră relaxată și distractivă. În plus, NAO a reușit să atragă atenția comunităților de dezvoltatori de programe software, care au afirmat ca acesta este o adevarată și impresionantă putere de calcul, un mediu potrivit pentru elaborarea aplicațiilor [1].

1.2 CARACTERISTICI HARDWARE

În vederea dezvoltării acestui robot, inginerii de la Aldebaran Robotics s-au bazat pe conceptele mecatronicii, domeniu tehnic ce se află la intersecția dintre electronică, mecanică și informatică.

În cele ce urmează vor fi prezentate detaliile tehnice ale robotului conform cu foile sale de catalog:

- **ALIMENTAREA**

Robotul este echipat cu o baterie de tipul Lithium-Ion, ce oferă o energie de 48,6 Wh și o autonomie de până la 60 de minute, daca este utilizat în mod activ sau 90 de minute, utilizat în mod normal. Încărcarea bateriei se face în aproximativ 3 ore, la un curent de 1,8 A.

- **CONSTRUCȚIA**

Dimensiunile robotului sunt 574 mm – înălțime, 275 mm – lățime, 311 mm – grosime, iar greutatea acestuia este de 5,4 kg. Aceste mărimi bine determinate influențează în mod direct performanțele de mișcare și susținere ale robotului.

Materialul de fabricație este o combinație între PC-ABS și Poliamida-66 (PA-66). Policarbonatul-ABS este unul din materialele cel mai des utilizate în industrie datorită flexibilității date de ABS, iar Poliamida-66 este un termoplastic versatil folosit în inginerie.

- **UNITĂȚILE DE COMANDĂ ȘI CONTROL**

Robotul NAO are două procesoare principale: unul localizat la nivelul capului, celălalt la nivelul toracelui.

Prima UCP, cea localizată la nivelul capului, este constituită dintr-un procesor Intel cu arhitectură x86, de tipul ATOM Z530 și are o memorie cache de 512KB. Viteza ceasului este de 1,6 GHz, iar setul de instrucțiuni este de 32 de biți, microcontrolerul fiind compatibil cu Intel System Controller Hub – Intel® SCH, componentă ce combină funcționalitățile plăcii grafice, ale interfeței procesorului și ale controlerului de memorie cu extensiile de intrare/ieșire, în vederea unui consum cât mai mic de energie [4].

ATOM Z530 este un procesor puternic, cu un singur nucleu, destinat în principal dispozitivelor mobile. Raportul performanță – preț este unul foarte bun. Avantajele importante ale acestuia sunt consumul de energie redus și faptul că suportă virtualizarea aplicațiilor. De asemenea, acesta este bazat pe microarhitectura Bonnell, fiind astfel capabil să execute până la maxim două instrucțiuni pe ciclu de ceas. Acesta face translația instrucțiunilor complexe (de tip CISC) în operații interne, simple (de tip RISC) înainte de execuție [4]. În figura următoare este prezentată diagrama procesorului, împreună cu toate perifericele compatibile cu acesta:

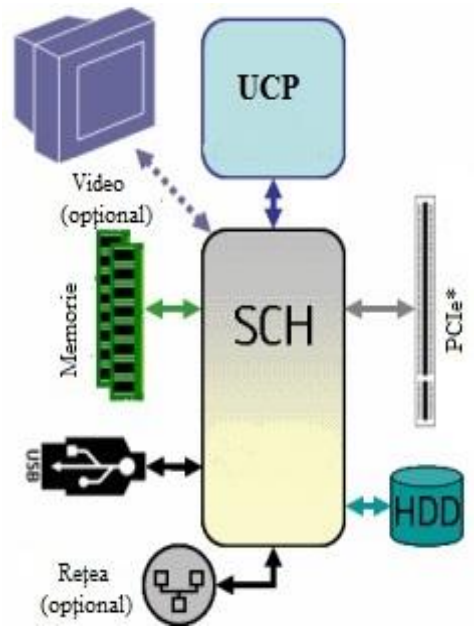


Figura 1.2.1 Diagrama ATOM Z530 [5]

Cea de-a doua UCP, cea de la nivelul toracelui, este de tipul ARM7TDMI și controlează actuatoarele, distribuind informația de control tuturor microcontrolerelor locale de la nivelul modulelor de actuatoare. ARM7TDMI este un microcontroler cu setul de instrucțiuni pe 32 de biți, de tipul RISC, oferind performanțe ridicate pentru un consum mic de putere.

Microcontrolerele locale de la nivelul modulelor de actuatoare sunt de tipul Microchip 16-biți dsPIC, comunicația între acestea și cea de-a doua UCP făcându-se prin intermediul a două magistrale de tipul RS485 (de 460Kbps).

- **SENZORI**[1]

Senzorii sunt liantul între un robot umanoid și lumea exterioară, fiind dispozitive ce măsoară atribute ale mediului înconjurător. Alături de celelalte două primitive, planificare și control, senzorii joacă un rol important în paradigmele roboticii [2]. Aceștia pot fi clasificați după tipul informației măsurate oferite la ieșire, astfel :

Exteroreceptivi – senzori de contact, video și audio.

Prioreceptivi – senzori care măsoară poziția, orientarea și viteza robotului.

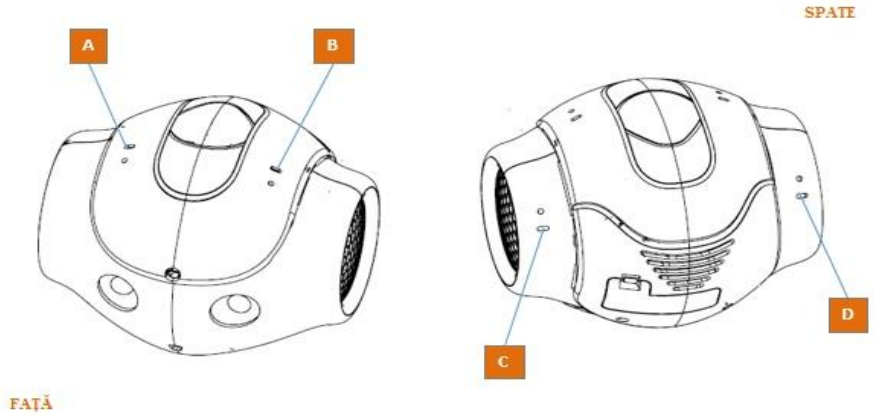
În tabelul de mai jos sunt enumerați toți senzorii prezenți în arhitectura hardware a robotului NAO. Aceștia vor fi pe scurt detaliați în cele ce urmează.

Tipul senzorului	Nr. de senzori pe NAO
Audio - difuzor	2
Audio - microfon	4
Video - cameră	2
Senzori de infraroșu (IR)	2
Rezistențe sensibile la forță	8
Senzori ultrasonici	2
Girometru	1
Accelerometru	1
Senzori tactili și de contact	8
Senzori de poziție	36

Tabel 1.2-1 Lista senzorilor de pe robotul NAO

Audio

Senzorii audio sunt plasați la nivelul capului, cele două difuzoare în laterale, iar cele patru microfoane sunt distribuite uniform ca în figura de mai jos:



7

Figura 1.2.2 Distribuția microfoanelor [1]

[1] Senzor, în lucrarea de față, este folosit în sens larg, incluzând și traductoarele prezente în arhitectura robotului (boxe, microfon).

Domeniul de frecvențe al microfoanelor este între 150 Hz și 12 kHz, pe când cel al difuzoarelor se delimitează la valoarea superioară de 20 kHz.

Video

NAO are două camere video, cu dublă funcționalitate: înregistrare și captare de imagini ce se regăsesc tot la nivelul capului, integrate sub forma unor ochi. Această poziționare foarte bine aleasă oferă un aspect extrem de prietenos feței robotului. Rezoluția oferită este de 1,22 MegaPixeli, iar formatul imaginilor captate este de tipul PNG. În cazul transmiterii video în timp real, calitatea imaginii nu este una extraordinar de bună datorită rezoluției mici, dar și vitezei de transmisiune dată de protocolul de comunicație utilizat.

Senzori de infraroșu

Sunt prezenți doi senzori de acest tip, poziționați în centrul așa-zișilor ochi. Fiecare dintre ei prezintă un emițător și un receptor, ce funcționază pe o lungime de undă de 940 nm, la un unghi de +/- 60°. Aceștia sunt utili pentru aplicațiile de comunicație între doi sau mai mulți roboți NAO, prin intermediul senzorilor putându-se face schimb de informații.

Rezistențe sensibile la forță

Acest tip de senzori au la bază un polimer conductiv care își modifică rezistența într-o maniera predictibilă, funcție de forța aplicată pe suprafața sa. Aceștia sunt utilizați pentru a crea zone sensibile la presiune. Robotul NAO are opt asemenea senzori, localizați câte patru pe fiecare talpă și sunt utili pentru aflarea stării de echilibru la un moment dat, calculând centrul de greutate pe fiecare picior. Domeniul de forțe este între 0 și 110 Newtoni, iar valoarea returnată de aplicația corespunzătoare este dată în kilograme.

Senzori ultrasonici

Senzorii ultrasonici sau sonari sunt constituiți din câte un emițător și un receptor ce funcționează la frecvența de 40kHz. Poziționarea acestora pe pieptul robotului evidențiază utilitatea lor, și anume detectarea unor obiecte aflate în față, la o anumită distanță. Conul de emisie este de 60°, iar domeniul de detecție a obstacolelor este între 0,25m și 2,55 m, sub 0,25m nu poate fi returnată nicio informație legată de distanță, robotul știind doar de existența obiectului.

Girometrul și Accelerometrul

Aceste două tipuri de senzori constituie unitatea inerțială a robotului, oferind informații vitale despre poziția în spațiu a acestuia în mers, mai exact despre stabilitatea lui. Girometrul măsoară orientarea (rad/s), bazată pe aflarea vitezei unghiulare date de 2 axe spațiale, iar accelerometrul măsoară accelerația proprie sau forța g(m/s^2).

Senzorii tactili și de contact

Se numesc senzori de contact deoarece descriu dacă ceva sau cineva atinge robotul. Aceștia pot fi enumerați astfel: butonul de pe piept, senzorii tactili de pe cap și de pe mâini și senzorii pe post de amortizoare de la nivelul picioarelor. Butonul de pe piept are diverse funcționalități de bază, cum ar fi pornirea robotului, aflarea adresei IP etc. Fiecărui senzor îi pot fi atribuite diverse aplicații, funcție de dorința utilizatorului.

Senzori de poziție

Deoarece poziția robotului este o caracteristică extrem de importantă, pe lângă senzorii deja descriși mai sus, robotul NAO mai prezintă și senzori expliciți de poziție, numiți encodere de rotație magnetice (ERM). Funcționarea acestora se bazează pe Efectul Hall, mișcările efectuate sunt resimțite prin străpungerea câmpului magnetic, ce determină o diferență de potențial prelucrată ulterior astfel încât să se obțină informații despre localizarea în spațiu.

LED-uri

NAO are instalat pe el o multitudine de LED-uri, cu diverse scopuri. Unele din ele vizează notificarea unor alerte, altele au scop animarea robotului. În tabelul ce urmează sunt enumerate LED-urile existente pe robot.

Localizare	Cantitate	Descriere
Pe cap	1 x 12	16 nivele de Albastru
Ochi	2 x 8	Toate culorile pe baza RGB
Urechi	2 x 10	16 nivele de Albastru
Butonul de pe piept	1 x 1	Toate culorile pe baza RGB
Picioare	2 x 1	Toate culorile pe baza RGB

Tabel 1.2-2 LED-urile de pe robotul NAO

- **ACTUATOARE**

Actuatoarele sunt motoarele responsabile cu mișcarea robotului.

Roboții umanoizi sunt construiți astfel încât să imite corpul uman, așadar actuatoarele se folosesc pe post de mușchi și încheieturi, dar au o structura diferită față de cea umană. Actuatoarele convertesc energia electrică în mișcare fizică, marea majoritate producând fie mișcare de rotație, fie liniară. Motoarele de curent continuu, cunoscute în domeniu ca motoare DC, pot fi privite ca fiind actuatoare. Robotul NAO deține motoare DC cu perii de carbon, deoarece au avantajul unui cost inițial mai mic și au viteza simplu de controlat. Motoarele de acet tip generează cuplu direct din sursa de alimentare, folosind comutație internă, magneți staționari și electromagneți rotativi.

Funcție de viteza de rotație și cuplul dezvoltat când motorul este blocat (cuplul maxim suportat), motoarele robotului NAO sunt clasificate în trei categorii astfel:

- Tipul 1: 8300rpm[2] +/- 10%, cuplul maxim – 68mNm[3]
- Tipul 2: 8400rpm +/- 12%, cuplul maxim – 9,4 mNm
- Tipul 3: 10700rpm +/- 10%, cuplul maxim – 14,3 mNm

Se poate observa că motoarele de Tipul 1 sunt unele puternice, datorită cuplului maxim ce poate fi dezvoltat în blocaj, prin urmare acestea sunt folosite la picioarele robotului, fiind capabile sa-l mențină stabil în mers și pe loc. Celelalte componente, cum ar fi capul, coatele, umerii sunt utilizate mai des în activități ce implică mișcarea, de aceea pentru ele se folosesc motoare cu viteze de rotație mai mari (Tipul 2 și 3).

[2] rpm = rotații pe minut
[3] mNm = mili-Newton * metru

În figura următoare sunt prezentate pozițiile motoarelor în robot, specificându-se numele acestora în limba engleză, conform cu documentația oficială, iar între paranteze tipul.

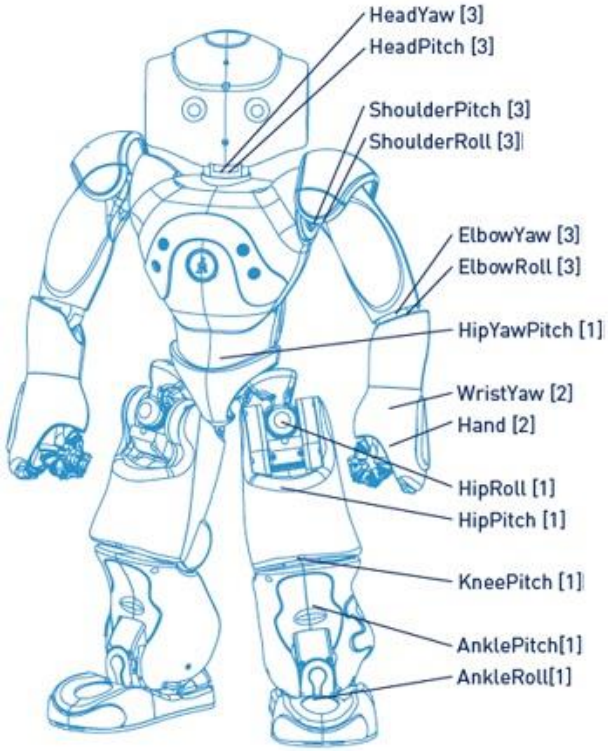


Figura 1.2.3 Motoarele robotului NAO [6]

- **GRADE DE LIBERTATE**

În mecanică, un grad de libertate exprimă fiecare dintre mărimile scalare independente (parametrii) necesare pentru determinarea univocă a stării unui sistem. Numărul minim de paramentrii prin care se poate preciza starea sistemului reprezintă numărul de "grade de libertate" al acestuia. Un punct material liber are trei grade de libertate din punct de vedere al poziției lui în spațiu, pentru determinarea acesteia fiind necesare trei mărimi reale (ex. coordonatele carteziene x,y și z). Fiecare dependență (legătură) bilateral impusă scade numărul gradelor de libertate cu o unitate[6].

NAO are 25 de grade de libertate, 11 pentru partea inferioară ce include picioarele și pelvisul și 14 pentru partea superioară formată din trunchi, mâini și cap. În tabelul 1.2 sunt precizate componentele robotului ce dețin grade de libertate și numărul acestora.

Componentă	Grade de libertate
Cap	2
Braț (x2)	5
Pelvis	1
Picior (x2)	5
Mână (x2)	1

Tabel 1.2-3 Gradele de libertate ale robotului NAO

- **CONECTIVITATE**

Robotul NAO suportă conexiuni prin două protocoale de comunicație arhicunoscute, Ethernet și Wi-Fi.

Pentru Ethernet, robotul are la nivelul capului, o mufă de tipul RJ-45, compatibilă cu 3 tipuri de adaptări: 10/100/1000 base T (viteze de 10Mbps, 1000Mbps sau 1 Gbps). Conexiunea prin cablu este cel mai des utilizată pentru setarea conexiunii prin WiFi. Astfel, după conectarea cablului trebuie apăsat butonul de pe pieptul lui NAO. Acesta va raspunde, în limba engleză, cu adresa IP care i-a fost alocată.Accesând dintr-un navigator de Internet, adresa IP aflată și introducând credențialele corecte se va deschide o pagină web, special creată pentru controlul sumar al robotului. O dată creată această conexiune, cablul poate fi scos, NAO trecând pe protocolul WiFi automat, nefiind nevoie de alte setări. Dacă rețeaua nu este modificată, la următoarea pornire a robotului nu va mai fi nevoie de stabilirea conexiunii WiFi prin Ethernet, ci se va conecta direct, păstrând aceeași adresă IP alocată anterior.

Conexiunea prin WiFi este cel mai des utilizată pentru comunicația între utilitarul de dezvoltare a aplicațiilor pentru robot, denumit Choregraphe, și NAO, el fiind compatibil cu standadrul IEEE 802.11 b/g/n.

Figura de mai jos surprinde întreaga arhitectură hardware, din puncte de vedere electronic, fiind evidențiate legăturile între toate componetele prezentate mai sus.

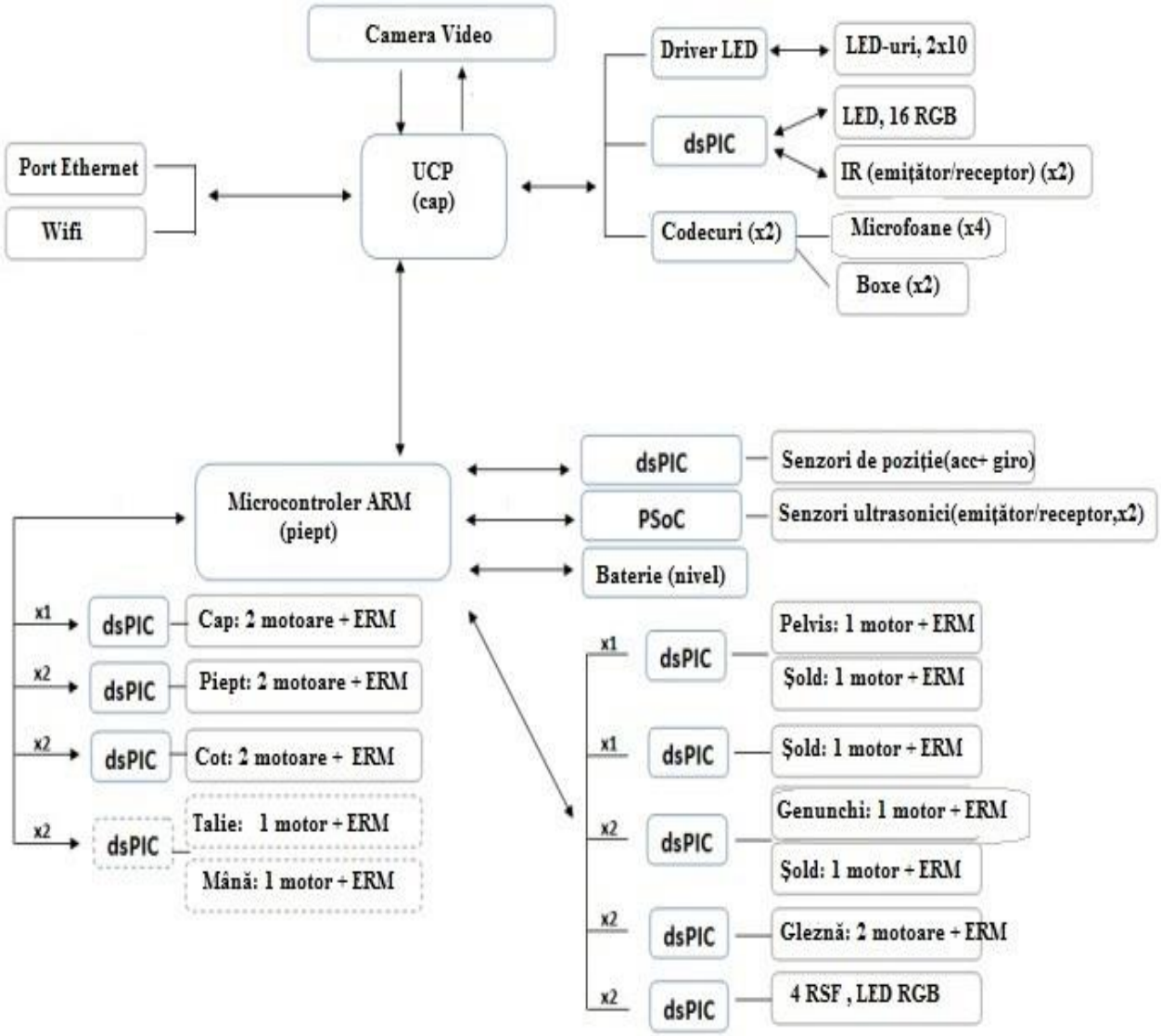


Figura 1.2.4 Arhitectura hardware d.p.d.v. electronic a robotului NAO

1.3 CARACTERISTICI SOFTWARE

Sistemul de operare al robotului se numește OpenNAO, ce rulează pe un nucleu de Linux și face parte din arhitectura software NAOqi, cea care dă viață robotului. Aceasta e o versiune integrată, special dezvoltată pentru roboții concepuți de firma Aldebaran. Ea asigură și rulează un număr mare de programe și librării. De asemenea, NAOqi este un sistem modular și distribuit capabil să lucreze cu un număr variabil de fișiere executabile, funcție de alegerile utilizatorului, arhitectura fiind una bazată pe evenimente.

Avantajele unui sistem distribuit sunt multe. Utilizatorul are posibilitatea de a rula comportamente ale robotului atât local cât și de la distanță. Funcționalitățile robotului, cum ar fi mișcarea, vederea, vorbirea etc., pot fi rulate pe robot în același fișier executabil sau în fișiere de sine stătătoare ce interacționează cu alte module de pe alte calculatoare. Dezvoltarea aplicațiilor este mult mai ușoară într-un mediu distribuit, deoarece același cod poate fi compilat pe platforme diferite. De asemenea, un astfel de sistem permite utilizatorului să vizualizeze variabilele și programele ce rulează pe orice robot, cu ajutorul programelor de interfațare.

Funcționalitățile arhitecturii NAOqi sunt următoarele:

- Suportă scrierea programelor în diferite limbaje. Utilizatorul poate controla robotul folosind limbaje de programare precum: C++, Python și Java;
- Execuția metodelor se poate face paralel, secvențial sau prin apeluri de evenimente;
- Managementul proceselor;
- Modularitatea - utilizatorul poate alege între a compila o aplicație ca o librărie dinamică sau ca un fișier executabil fără a fi nevoit să schimbe codul sursă;
- Suportă mai multe platforme, cum ar fi Linux, Windows sau MAC OS;
- Managementul interfețelor pentru programarea aplicațiilor (cunoscute ca API);
- Managementul memoriei partajate;

Din punct de vedere structural, arhitectura NAOqi poate fi divizată în trei părți:

- Sistemul de operare – NAOqi SO sau OpenNAO – despre acesta s-au precizat detalii la începutul subcapitolului;
- Biblioteca NAOqi – este divizată în obiecte ce conțin acțiuni pe care robotul le poate face. Avantajul acestei biblioteci este dat de nivelul de abstractizare, utilizatorul fiind capabil să programeze robotul fără acces direct la aceasta;
- Managerul de comunicații al dispozitivelor (DCM – Device Communication Manager) – este un modul văzut ca o bibliotecă, rolul său este de a controla comunicația cu dispozitivele electronice din robot, cu excepția senzorilor audio și a camerei video. DCM poate fi văzut ca legătura între nivelul software superior, compus din alte module și nivelul software inferior, care e integrat în plăcile electronice. Acesta preia informații de la dispozitivele electronice prin intermediul celei de-a doua UCP și de asemenea, accesează dispozitive localizate la nivelul capului prin intermediul magistralelor de date.

În figura următoare este prezentată legatura,dată de DCM între nivelul software superior și inferior:

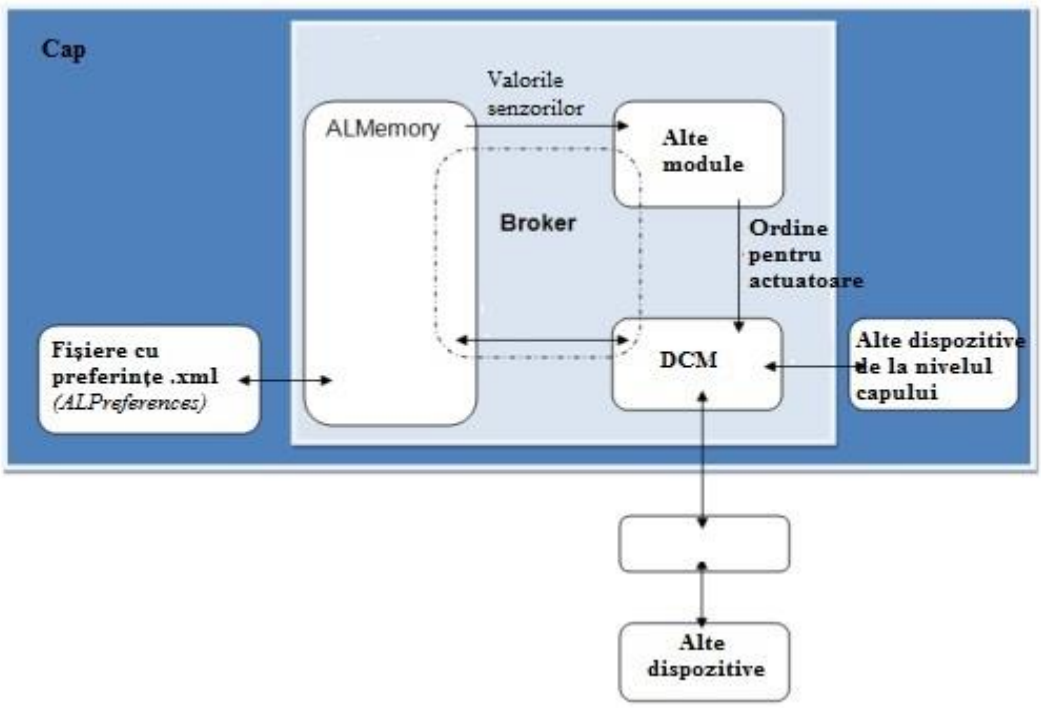


Figura 1.2.5 Legătura între nivelul software superior și cel inferior [6]

Arhitectura NAOqi este una orientată pe obiecte. Așadar există trei tipuri principale de obiecte:

Broker (agent) – rolul său este de a expune modulele întregii arhitecturi. Pentru a fi accesibil, un modul este obligatoriu să fie legat de un broker. Agenții controlează rețeaua de comunicație. Un broker poate fi copil pentru un altul, formându-se astfel un arbore distribuit.

Modul – acesta conține metodele definite de utilizator.

Proxy[4]– este un obiect ce se comportă ca și modulul pe care îl reprezintă, de exemplu la crearea unui proxy către modului *ALMotion*, se va crea un obiect ce conține toate metodele modului.

La crearea unui modul, dezvoltatorul instanțiază un broker de care leagă apoi modulul său. Modulul își declară automat metodele legate. În orice moment, utilizatorul poate instanția un proxy prin specificarea numelui modulului dorit. Imediat ce proxy-ul este pregătit, metodele corespunzătoare modulului pot fi apelate, indiferent dacă modulul este local sau la distanță, sau dacă este scris în C++ sau în oricare limbaj de programare compatibil.

Nucleul sistemului NAOqi este determinat de trei module complexe, denumite *ALMemory*, *ALLogger* și *ALPreferences*. Ele sunt încărcate automat la pornirea robotului, *ALMemory* activează modul de partajare al memoriei. Modulele predefinite de Aldebaran și cele ale utilizatorului pot adăuga sau verifica variabile din *ALMemory*. Modulul *ALPreferences* face managementul tuturor caracteristicilor customizabile și inițializează fișiere XML. Fiecare modul poate folosi *ALPreferences* pentru a citi sau scrie atribute sau pentru a le stoca în modulul *ALMemory*.

Conectarea la robotul NAO, în vederea dezvoltării aplicațiilor, se poate face prin două modalități, fie prin intermediul utilitarului dezvoltat de Aldebaran, Coregraphe, fie de la distanță, de pe orice calculator, prin intermediul protocolului de comunicație SSH. Detalierea programului

[4] Pentru acest termen nu există un corespondent în limba română care să aibă înțelesul potrivit.

Coregraphe va fi făcută într-un capitol următor. Pentru cea de-a doua modalitate de conectare, este necesară instalarea unui program ce emulează un terminal - client pentru SSH. Aproape orice distribuție Linux, vine cu un client SSH instalat implicit.

CAPITOLUL 2

RECUNOAȘTEREA AUTOMATĂ A VORBIRII

Procesul de Recunoaștere Automată a Vorbirii (RAV) poate fi definit ca transcrierea independentă și în timp real a unui semnal audio ce conține vorbire într-o succesiune de cuvinte. Semnalul audio se dorește a fi o secvență de cuvinte rostite în mod cât mai natural. Un sistem RAV susține interacțiunea om - mașină. Un asemenea sistem ideal ar putea fi capabil să reproducă vorbele unei persoane la un procentaj maxim, oferind certitudine și siguranță astfel încât să nu mai fie nevoie de alte interacțiuni, decat aceea prin vorbire. Desigur, acesta este idealul absolut, în realitate existând o multitudine de factori ce influențează comportamentul unui sistem de recunoaștere automată a vorbirii cum ar fi zgomotul de fundal, tipul vorbitorului, particularitățile limbii vorbite etc. .

Cercetările din ultimele decenii au dus la dezvoltarea impresionantă a sistemelor de tip RAV, astfel că în prezent omul se poate folosi de acestea prin intermediul dispozitivelor inteligente cu scopul de a facilita activități precum apelarea unei persoane din agenda telefonică, deschiderea unei aplicații ce rulează pe dispozitivul respectiv, comunicarea cu un robot umanoid sau cu automobile de ultimă generație, deschiderea ușilor sau aprinderea luminilor într-o casă inovativă etc. .

Dat fiind scopul principal al unui sistem de recunoaștere vocală, acela de a transforma într-un text un semnal audio ce conține vorbire, se pot enumera principalii factori ce influențează calitatea răspunsului sistemului:

- Mediul de proveniență al semnalului audio (zgomotos/silențios);
- Vorbitor cunoscut / necunoscut sistemului;
- Caracteristicile vorbitorului: accent, dialect, rapiditatea pronunției;
- Stilul vorbirii: cuvinte izolate, vorbire continuă / spontană;
- Morfologia complexă a limbii utilizate;
- Dimensiunea vocabularului;
- Incertitudinea lingvistică a discursului;

2.1 VORBIREA

Utilizarea limbajului vorbit este principala modalitate de comunicarea între oameni și chiar între oameni și mașini, după cum va fi prezentat în această lucrare.

Vorbirea se produce datorită unui flux de aer ce pornește din plămâni sau diafragmă, trece prin laringe unde, prin intermediul corzilor vocale, este modulat. Această etapă se numește fonație și determină înălțimea și tonul semnalului vocal. La nivelul laringelui, există patru corzi vocale, două superioare și două inferioare, între care se află o deschizătură (glotă) ce diferă ca dimensiune la femei și la barbați. Sistemul nervos central este cel ce stimulează și controlează activitatea corzilor. Mai departe, faringele dirijează direcția fluxului de aer spre cavitatea bucală și cea nazală, simultan sau pe rând, producându-se astfel articularea.

Fiziologia aparatului vocal determină caracteristicile spectrului vorbirii. Oscilația glotei determină o frecvență fundamentală (f_0) și o serie de armonici ale căror valori sunt multiplii ai f_0 . Poziția pe axă a frecvenței fundamentale este dependent de vorbitor și responsabilă, în mod direct, de înălțimea vocii.

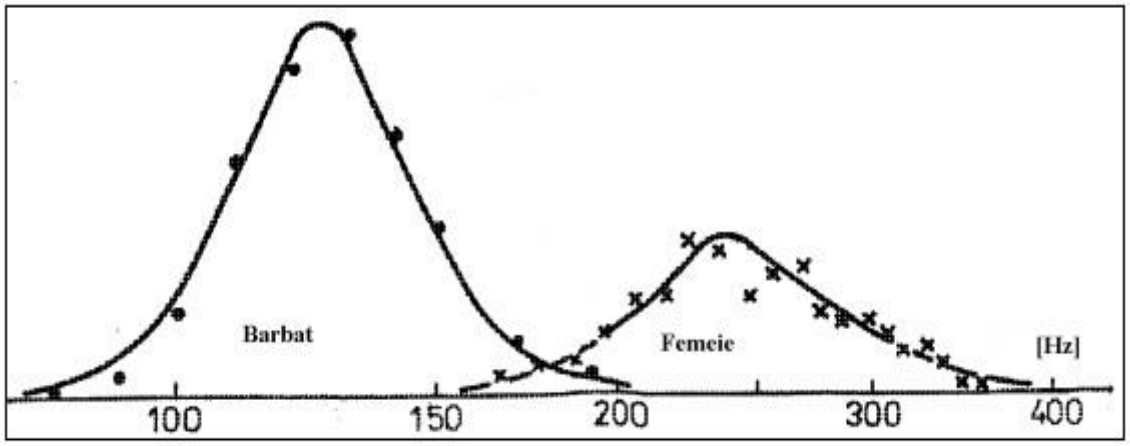


Figura 2.1.1 Distribuția frecvențelor vocii la barbat și femeie [8]

Cea mai mică unitate sonoră a limbii, cu funcția de a diferenția cuvintele între ele, precum și formele gramaticale ale aceluiași cuvânt se numește *fonem*. Prin modificarea unui fonem al unui cuvânt, se generează fie un cuvânt inexistent, dar perceput ca fiind diferit de către vorbitorii limbii, fie un cuvânt cu alt sens. Acestea, în special vocalele, au regiuni caracteristice de energie ridicată în cadrul spectrului, localizarea lor ajutând atât pentru identificarea fonemelor cât și a vorbitorului. Cea mai mare parte din energia semnalului vocal se regăsește între 0 și 4 kHz.

În majoritatea limbilor din lume, fonemle sunt împărțite în două mari categorii:

- Consoanele – produse prin articulația cu ajutorul obstacolelor formate de componentele cavității bucale (dinți, limbă) în calea fluxului de aer;
- Vocalele – produse tot prin articulație, dar de această data fără obstacole.

De asemenea, și sunetele pot fi împărțite în subgrupuri. În cazul limbii române există următoarea clasificare a sunetelor funcție de articularea acestora [9]:

Vocalele

Ele se disting prin înălțimea, dată de frecvența fundamentală, și prin timbru, reprezentat prin conținutul seriei de armonici.

În funcție de unghiul de deschidere a maxilarelor (apertura):

- Vocale deschise: a.
- Vocale medii (semideschise): e, ă, o.
- Vocale închise: i, î(â), u.

În funcție de localizarea lor, mai exact de locul de articulare (punctul din cavitatea bucală în care se creează spațiul optim de rezonanță, realizat prin diferitele poziții pe care le ia mușchiul lingual):

- Vocale anterioare: e, i.
- Vocale centrale (neutre) : a, ă, î(â).
- Vocale posterioare: o, u.

În funcție de modul în care sunt însoțite sau de rotunjirea buzelor:

- Vocale rotunjite: o, u.
- Vocale nerotunjite: a, e, i, ă, î(â).

Consoanele

Acestea se pot clasifica în două categorii principale, funcție de apropierea de vocale:

- Consoanele nesonante - nu au caracteristici asemănătoare cu cele ale vocalelor (p, b, f, v, t, d, s, z, ț, ș, j, c, g, h).
- Consoanele sonante – este o clasă distinctă, deoarece prezintă caracteristici comune atât cu vocalele, cât și cu consoanele. Astfel, la articularea lor predomină tonurile muzicale, zgomotele specifice consoanelor fiind mai slab (m, n, l, r).

2.2 ARHITECTURA SISTEMULUI RAV

Recunoașterea automată a vorbirii este unul dintre primele domenii în care tehnicile de modelare statistică (unde modelele se crează pe baza unui număr mare de date) s-au impus ca și standard. Platforma statistică pentru recunoașterea automată a fost creată și dezvoltată de echipe de cercetare ale companiilor IBM și AT&T.

Procesul de transformare a vorbirii în text poate fi formulat, din punct de vedere statistic, astfel: *Care este cea mai probabilă secvență de cuvinte W în limba L, dat fiind mesajul vorbit X?[10]*

Secvența X este o secvență de eșantioane consecutive ale semnalului audio:

$$X = x_1, x_2, \dots, x_t \qquad (2.1)$$

Secvența W reprezintă o secvență de cuvinte, iar w_i este un cuvânt individual:

$$W = w_1, w_2, \dots, w_n \qquad (2.2)$$

În acest moment, rezultatul poate fi cuantificat folosind funcția *argmax*, care selectează argumentul ce maximizează probabilitatea secvenței de cuvinte. Așadar, cea mai probabilă secvență ($\widehat{W}$) este cea cu cea mai mare probabilitate apriori (P(W|X)), dacă la intrare avem secvența X):

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

Folosind criteriul de decizie Bayes, ecuația (2.3) devine:

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

În ecuația (2.4), termenul P(X) este probabilitatea secvenței de intrare, independent de secvența de cuvinte W, deci poate fi neglijat. Relația se poate rescrie astfel:

$$\widehat{W} = \arg\max_w P(W|X)P(W) \quad (2.5)$$

Problema inițială (găsirea secvenței de cuvinte pe baza mesajului audio ce conține vorbire) este acum împărțită în două sub-probleme mai simple:

1) Estimarea probabilității apriori a secvenței de cuvinte P(W)
2) Estimarea probabilității mesajului vorbit dată fiind secvența de cuvinte pronunțată P(W|X).

Primul factor pote fi estimat utilizând un *model de limbă*, iar ce-l de-al doilea poate fi estimat cu ajutorul unui *model acustic.* Cele două modele pot fi construite independent, dar vor fi folosite împreună pentru a decoda un mesaj vorbit conform ecuației (2.5).

Arhitectura generală a unui sistem de recunoaștere automată a vorbirii este prezentată în figura următoare:

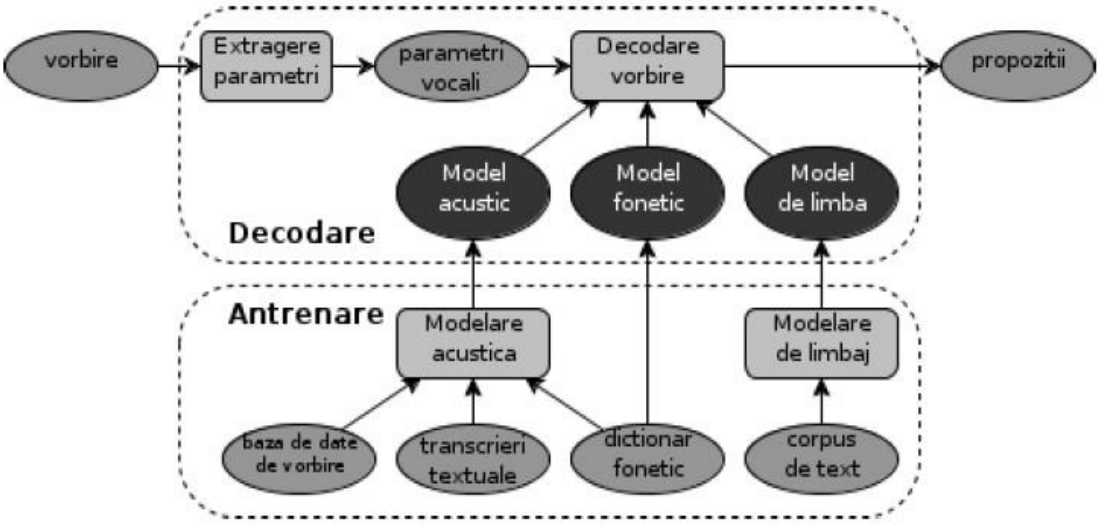


Figura 2.2.1 Arhitectura sistemului RAV [10]

Din figura 2.2.1 reies două lucruri esenţiale în ceea ce priveşte procesul de recunoaştere a vorbirii:

1) recunoaşterea se face utilizând o serie de parametri vocali extraşi din semnalul vocal (nu direct, utilizând mesajul vorbit)

2) recunoaşterea se face pe baza unor modele (acustic, fonetic şi lingvistic) dezvoltate în prealabil.

Modelul acustic are rolul de a estima probabilitatea unui mesaj vorbit, dată fiind o succesiune de cuvinte. În sistemele de recunoaștere a vorbirii continue actuale modelul acustic nu folosește cuvinte ca unităţi acustice de bază pentru că fiecare sarcină de RAV are un vocabular de cuvinte diferit pentru care nu există modele deja antrenate şi nici date de antrenare disponibile, iar numărul de cuvinte diferite dintr-o limbă este prea mare. În locul cuvintelor, se utilizează unităţi acustice de bază sub-lexicale, de exemplu foneme, sau, mai recent, chiar unităţi sub-fonemice numite senone. Prin urmare, modelul acustic este format dintr-un set de modele pentru foneme (sau senone) care se conectează în timpul procesului de decodare pentru a forma modele pentru cuvinte şi apoi modele pentru succesiuni de cuvinte. Acestea sunt utilizate în final pentru a estima probabilitatea ca mesajul rostit de vorbitor să fie format dintr-o succesiune sau alta de cuvinte. Experienţa a arătat că această abordare generativă pentru modelele acustice poate fi foarte bine implementată utilizând modele Markov ascunse (Hidden Markov Models - HMM).

Modelul de limbă este utilizat în timpul decodării pentru a estima probabilităţile tuturor secvenţelor de cuvinte din spaţiul de căutare. În general, rolul unui model de limbă este de a estima probabilitatea ca o secvenţă de cuvinte să fie o propoziţie validă a limbii. Probabilitatea acestor secvenţe de cuvinte ajută foarte mult modelul acustic în procesul de decizie. Rolul modelului de limbă este acela de a asigna o probabilitate mult mai mare secvenţei de cuvinte cu sens logic şi, prin urmare, de a ajuta sistemul de RAV să decidă în favoarea acesteia.

Figura 2.2.1 ilustrează, de asemenea, procesele implicate în dezvoltarea unui sistem de RAV, dar şi resursele necesare creării modelelor acustice, lingvistice şi fonetice. Modelul acustic se construieşte strict pe baza unui set de clipuri audio înregistrate asociate cu transcrierea textuală a mesajelor vorbite şi a unui dicţionar fonetic ce cuprinde toate cuvintele din respectiva transcriere textuală.

3.2.2 RESURSELE FONETICE, ACUSTICE ȘI LINGVISTICE

Din figura cu arhitectura sistemului de recunoaștere vorbirii se pot observa resursele necesare antrenării celor trei modele amintite anterior (acustic, fonetic și lingvistic).

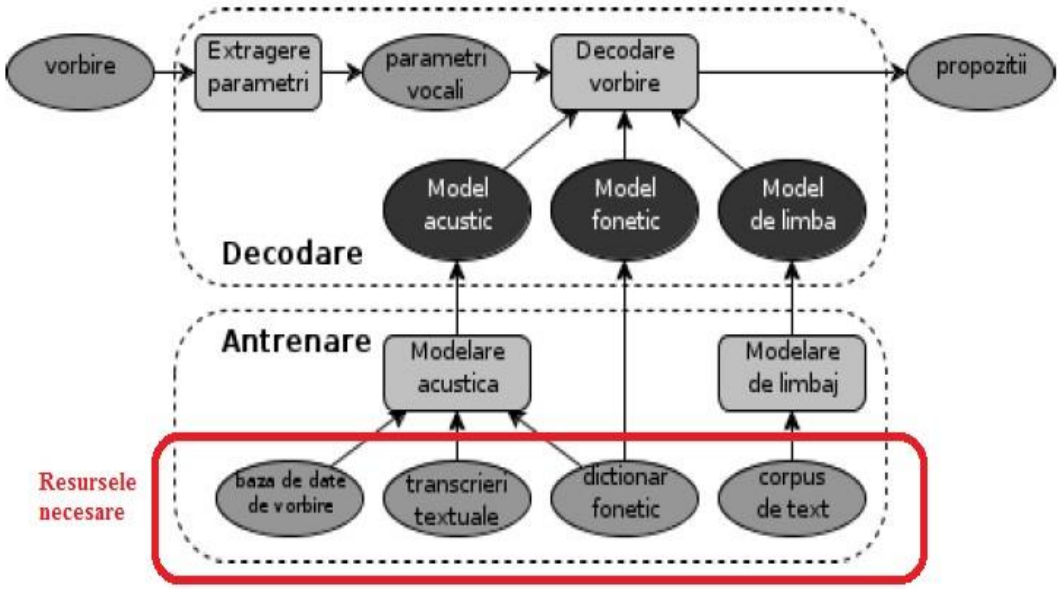


Figura 2.2.2 Resursele necesare necesare unui sistem RV [10]

- *Baza de date de vorbire* - este constituită de o multitudine de înregistrări audio ce conțin vorbire. În majoritatea cazurilor, aceste clipuri sunt eșantionate cu frecvența de 26kHz, dimensiunea eșantioanelor fiind de 16 biți, iar formatul fișierelor .wav. Aceste caracteristici limitărilea pe care un utilizator ce dorește folosirea unui sistem RAV trebuie să le respecte, în vederea obținerii unui răspuns optim.

Componentele principale ale acestei baze de date sunt:

- un set de clipuri audio;
- un set de fișiere text ce conțin transcrierile celor rostite în clipuri, precum și marcaje temporale pentru fiecare cuvânt/fonem;
- informații suplimentare ce vizează domeniul vorbirii, identitatea vorbitorului;

Aceasta este utilizată în antrenarea unui sistem de recunoaștere a vorbirii, fiind un elemnt cheie care influențează în mod direct performanțele sistemului: rata de eroare și viteza de răspuns. Cele mai importante caracteristici ale bazei de date care determină calitatea acesteia sunt următoarele:

- dimensiunea: numărul de ore înregistrare și numărul de vorbitori;
- stilul vorbirii: cuvinte izolate, vorbire continuă citită, vorbire conversațională;

- variabilitatea: calitatea înregistrărilor, zgomotul de fundal, diversitatea vorbitorilor;

- *Dicționarul fonetic* – acesta este un instrument lingvistic care specifică modul în care trebuie pronunțate cuvintele unei limbi. Mai exact, acesta face corespondența între forma literară a cuvintelor și forma fonetică (o succesiune de foneme).

În sistemul de RAV, dicționarul fonetic este cel care face legătura între modelul acustic (care modelează producerea sunetelor specific limbii vorbite) și modelul de limbă (care modelează maniera în care se succed cuvintele limbii).

Primul pas în construcția unui dictionar fonetic constă în crearea formei fonetice ale cuvintelor vocabularului. Referitor la cuvintele din limba română, regulile de fonetizare au fost studiate sumar în literature de specialitate. De aceea, pentru procesul de construcție al dicționarului, membrii echipei SpeeD au stabilit anumite reguli de fonetizare, cât și excepțiile acestora. Totuși, din fericire, pentru limba română, diferențele între forma scrisă si pronunția unui cuvânt sunt relativ mici. În general, fiecărei litere îi corespunde întotdeauna aceluiași fonem. Există doar câteva cazuri în care aceeași literă se pronunță altfel, în funcție de contextul în care apare. Odată stabilit setul de reguli, fonetizarea se poate realiza și automat, prin intermediul aplicațiilor software care implementează setul de reguli stabilit a priori. În tabelul următor sunt enumerate clasele de ambiguitate în fonetizarea cuvintelor din limba română:

Litera	Fonemul	Exemple
e	e	cer, rest, rece
	e_1	deal, te-am, ceainic
	i_3e	eu, el, eram, erai
i	i	fir, citit, prins
	i_1	rupi, deci
	i_3	iarbă, doi, fier
o	o	potrivit
	o_1	voal, soare
u	u	sud, furtun
	w	nouă
c	k	casă, sac
	k_1	cer, circ
	k_2	chiar, ochelari
g	g	gară, olog
	g_1	girafă
	g_2	unghi, ghem
h	h	hartă
	mut	ochi, unghi
x	ks	pix, fix
	gz	exemplu, examen

Tabel 2.2-1 Clase de ambiguitate în fonetizarea limbii române [10]

- *Corpusul de text* – această resursă este necesară în partea de dezvoltare a unui sistem de recunoaștere, mai précis la antrenarea modelului lingvistic.

Capacitatea de predicție a modelului de limbă depinde de dimensiunea corpusului de text folosit la antrenare, deoarece pe măsură ce numărul cuvintelor și secvențelor de cuvinte dintr-o limbă este mai mare, probabilitățile de apariție ale acestora sunt mai dificil de estimat cu acuratețe. Achiziția unui corpus de text de dimensiuni ridicate nu poate fi făcută automat. Cele mai utilizate metode de dezvoltare folosesc principiul Web-as-Corpus, care implică identificarea, descărcarea și procesarea materialelor text de pe Internet.

3.2.2 MODELAREA ACUSTICĂ

Modelul acustic poate fi supranumit nucleul unui sistem de recunoaștere a vorbirii. Acesta utilizează modele pentru foneme care sunt conectate, în timpul procesului de decodare, pentru a forma modele pentru cuvinte, apoi modele pentru secvențe de cuvinte.

- *Parametrii acustici*

Așa cum s-a menționat în introducerea capitolului curent, în sistemele de RAV se utilizează Modele Ascunse Markov (HMM), mai exact în etapa modelării acustice. Aceste modele Markov sunt automate probabilistice cu stări finite care pot fi combinate în mod ierarhic pentru a construi modele pentru secvențe de cuvinte din modele pentru unități de vorbire mai scurte cum sunt fonemele. Mai multe detalii despre acestea vor fi prezentate în paragraful intitulat *Platforma HMM.*

După cum este ilustrat în figura arhitecturii unui sistem de RAV, un bloc de extragere de parametrii este folosit pentru calculul unor vectori de coeficienți. Semnalul vocal este un semnal nestaționar, adică parametrii săi nu sunt constanți în timp. În consecință, analiza spectrală nu poate fi făcută pe întreg semnalul, ci pe cadre cvasi-staționare din acesta, de durată între 20ms-30ms. Semnalului initial, în domeniul timp, îi este aplicat o fereastră de tip Hamming, fiind astfel transformat într-o secvență de ferestre. Din acestea sunt extrași parametrii acustici.

Cei mai populari parametrii acustici sunt cei de tip cepstral-perceptual, mai exact parametrii Mel-cepstrali (Mel-Frequency Cepstrum Coefficients – MFCCs).

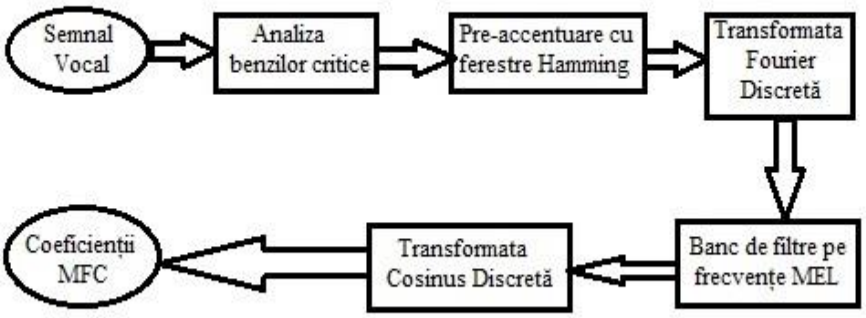


Figura 2.2.3 Schema bloc pentru analiza coeficienților MFCC

Figura 2.2.3 prezintă procedeul de obținere al coeficienților MFCC. Cum s-a menționat anterior, semnalul vocal este trecut print-o fereastră, apoi i se aplică o transformată Fourier discretă pentru a obține spectrul. Scara Mel de frecvențe simulează modul de percepție a frecvențelor în urechea umană. Bancul de filtere pe frecvențe Mel este divizat în benzi a căror lărgime crește de-a lungul axei o data cu un index. Fiecărei sub-benzi îi corespunde un filtru trece-bandă triunghiular. Spectrul semnalului în discuție este plasat conform scării Mel, iar logaritmul puterii este calculate în fiecare frecvență. Ultimul pas constă în aplicarea unei transformate cosinus discrete cu scopul de a obține o reprezentare nivelată și ortogonalizată. În final, rezultă câte șapte coeficienți cepstrali pentru fiecare vector Mel.

- *Platforma HMM*

Așa cum s-a specificat și mai sus, un HMM este un automat finit de stări format dintr-un set de stări conectate cu arce ce reprezintă tranzițiile. Secvența de stări este ascunsă, indisponibilă direct observatorului, de aceea întregul process Markov este considerat ascuns. Observatorul are acces doar la secvența de vectori de parametrii acustici.

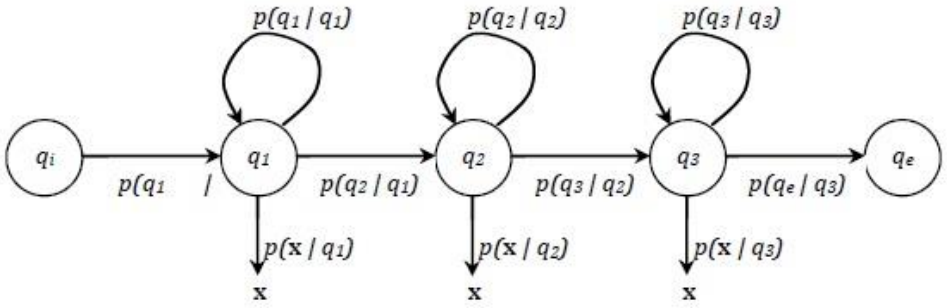


Figura 2.2.4 Reprezentarea unui HMM ca un automat de stări[10]

Figura 2.2.4 ilustrează un HMM ca un automat probabilist cu număr finit de stări, ale cărui stări inițiale și finale sunt non-emisive (nu duc spre alte stări). Un HMM poate fi caracterizat de următorii parametrii:

- un set de stări $Q = q_1 q_2 q_3 \ldots q_N$;
- un set de probabilități $A = a_1 a_3 \ldots a_N$ ce reprezintă probabilitățile de tranziție între stări;
- un set de observații probabilistice $B = b_i(x_t) = p(x_t | q_i)$ fiecare exprimă probabilitatea ca observația x_t să fi fost generate de starea I;

Deși definiţia unui HMM permite tranziţia din orice stare în alta mai puțin cele de la capete, în RAV modelele sunt create astfel încât să interzică tranziţiile arbitrare, deoarece este foarte important şi util ca modelul să ţină cont de caracterul secvenţial al producerii vorbirii. Așadar, impune constrângeri suplimentare privind tranziţiile înapoi sau salturile peste stări. Probabilităţile parametrilor generaţi de o stare *qi* sunt de obicei modelate de o funcţie densitate de probabilitate, care de cele mai ulte ori ia forma unei sume de k funcţii Gaussiene (Gaussian mixture model - GMM) în *d* dimensiuni.

Modelarea vorbirii prin intermediul HMM-urilor face două presupuneri importante [11]:

- proces Markov: se presupune că tranziţia între stările unui HMM este un proces Markov de ordin întâi în care probabilitatea traziţiei următoare depinde numai de starea curentă;
- independenţa observaţiilor: probabilitatea ca o stare să genereze un anumit vector de parametri este independentă de vectorii de parametri generaţi în prealabil.

Aceste două presupuneri pot conduce la un model al producerii vorbirii nerealist, însă ele sunt necesare pentru simplificările matematice şi computaţionale pe care le aduc.

Problemele utilizării platformei HMM sunt: evaluarea, decodarea şi estimarea, pentru fiecare dintre acestea existând algoritmi.

3.2.2 MODELAREA FONETICĂ

Modelul fonetic introduce și acesta restricții în procesul de decizie, deoarece nu toate combinațiile de foneme sunt posibile. Elementul cheie în modelarea fonetică este reprezentat de *Dicționarul fonetic*, prezentat în detaliu în paragrafele anterioare.

S-a menționat deja că pentru sistemele de recunoaștere vocală unitatea de modelare nu este cuvântul, ci o sub-unitate a sa, fonemul. Mai trebuie specificat faptul că există, în plus, alte criterii de care trebuie să se țină cont în alegerea unității de vorbire și anume [12]:

- Precizia - unitatea trebuie sa aibă realizarea acustică care apare în diferite contexte;
- Antrenabilitatea – numărul datelor folosite în estimarea parametrilor respectivei unități trebuie să fie sufficient de mare;
- Generalizarea – cu ajutorul unui set de bază trebuie să se poată construi orice cuvânt nou;

Se poate afirma ca fonemele îndeplinesc două dintre criteriile enumerate deoarece ele sunt și antrenabile și generalizabile. Antrenabile sunt datorită faptului că probabilitatea lor de apariție este mare, mai ales într-o bază de date de dimensiuni mari, iar generalizabile deoarece sunt independente de vocabularul sarcinii de recunoaștere. Totuși, pentru că oamenii nu pot pronunța fonemele identic în orice context, modelul fonetic nu mai este potrivit. Fonemele sunt produse dependent de cele din vecinătate din cauza fiziologiei aparatului uman de vorbire, fapt ce duce la o generalizare prea mare, ceea ce are drept consecință o scădere semnificativă a preciziei.

Există un model fonetic capabil să ia in considerare ambele foneme vecine, și anume modelul de tip trifonem. Principul acestuia este simplu de explicat: dacă două foneme identice se regăsesc în mesajul vorbit, în contexte diferite, ele sunt considerate trifoneme diferite. Folosirea acestui tip de model ar crește precizia, însă ar crește și numărul parametrilor ce trebuie estimați și antrenați. În prezent, pentru sisteme de RAV se folosesc așa zisele senone, ca unități de vorbire. Acestea au fost introduse de cercetătorii în domeniu, ca fiind sub-unități ale fonemelor, datorită compatibilității acestora cu cele trei criterii prezentate mai sus.

3.2.2 MODELUL DE LIMBĂ

Modelul de limbă este ultimul sub-bloc din arhitectura sistemului de RAV care va fi detaliat. Cum s-a mai spus deja, rolul lui este de a estima probabilitatea unei secvențe de cuvinte $\mathrm{W} = w_1, w_2, \dots, w_\mathrm{n}$ să fie o secvență validă pentru sarcina de recunoaștere. Probabilitatea secvenței de cuvinte poate fi descompusă astfel:

$$P(W) = P(w_1, w_2, \dots, w_\mathrm{n}) = P(w_1)P(w_2|w_1) \dots P(w_\mathrm{n}|w_1, w_2, \dots, w_{\mathrm{n}-1}) \quad (2.2.1)$$

Din ecuația 2.2.1 rezultă că problema estimării unei probabilități a secvenței de cuvinte W poate fi este împărțită în mai multe probleme de estimare a probabilității unui cuvânt relativ la secvența de cuvinte anterioară. Din cauza limitărilor impuse de eficiența computațională (a timpului de procesare) istoriile de cuvinte precendente nu pot include un număr indefinit de cuvinte, ci sunt limitate la ultimile *m* cuvinte. Așadar, se presupune ca doar un număr limitat de cuvinte afectează probabilitatea următorului cuvânt, ceea ce duce la folosirea modelului de limbă convențional de tip *n-gram,* model extrem de utilizat în domeniu. Factorul *m* este ales în funcție de cantitatea disponibilă de text pentru antrenare (este necesară o cantitate mare de text pentru a putea estima cu o precizie bună probabilitățile pentru istorii mai lungi). S-au introdus și așa-zisele modele de limbă de tip trigram, care iau in considerare doar ultimele două cuvinte pentru a-l prezice pe al treilea.

Modelul de limbă de tip n-gram se construiește prin estimarea probabilităților exprimate mai sus pe baza unor corpusuri de text de dimensiuni mari. În cazul unui model bigram, vor trebui estimate probabilitățile $P(w_\mathrm{j}|w_\mathrm{i})$ succesiv fiecărei perechi de cuvinte $(w_\mathrm{j}|w_\mathrm{i})$. Calcularea acestor probabilități este realizată cu ajutorul principiului probabilității maxime; se va număra de câte ori cuvântul w_i este urmat de cuvântul w_j comparativ cu alte cuvinte. Estimarea acestor probabilități necesită o cantitate mare de date de antrenare (de ordinul a câteva zeci de milioane de cuvinte).

Dezavantajul întâlnit la construcția modelelor de tip n-gram este dat de faptul că, indiferent de dimensiunea corpusului de antrenare vor exista n-grame ce nu vor apărea în acest text, însă pot apărea în textul de evaluare. Probabilitățile ce au fost estimate pe baza numărului de apariție a n-gramelor în corpusculi trebuie ajustate. Metodele ce țin de acest proces de ajustatare poartă numele de metode de netezire și se bazează pe extragerea unei părți din probabilitatea ce se alocă n-gramelor întâlnite la antrenare și redistribuirea acesteia n-gramelor necunoscute.[12]

2.3 EVALUAREA SISTEMULUI RAV

Evaluarea sistemului de recunoaştere a vorbirii este o etapă esenţială în dezvoltarea şi optimizarea acestuia. Aceasta îşi propune determinarea unui grad de corectitudine a sistemului, cât de adecvat este pentru a transcrie vorbire, în contextul unei sarcini de recunoaştere bine precizate. Evaluarea unui sistem de recunoaştere a vorbirii necesită existenţa unei baze de date de înregistrări audio etichetate, numită în continuare bază de date de evaluare. Pentru o evaluare corectă, baza de date de evaluare trebuie să reprezinte cât mai fidel sarcina de recunoaştere la care va fi supus sistemul.

În evaluarea întregului sistem de recunoaştere vocală, se iau în considerare și criteriile de evaluare ale modelului de limbă. Capacitatea de predicție a modelului de limbă se poate analiza măsurând probabilitatea acordată secvențelor de cuvinte din baza de date de evaluare. Un model de limbă adecvat, adaptat domeniului vorbirii, ar trebui să atribuie o probabilitate mare transcrierilor din baza de date de evaluare. Criteriul de performanță utilizat în acest caz este *perplexitatea*. Perplexitatea (*PPL*) este o mărime derivată din entropie (*H(pLM)* – mărime ce măsoară incertitudinea unei distribuții de probabilitate). În general, o valoare scăzută a acestei mărimi e corelată cu o mai bună funcționare a modelului de limbă, adică o capacitate de predicție a secvenței de cuvinte mai mare.

Un alt criteriu în evaluarea modelului de limbă este definit pe baza faptului că există șansa ca unele cuvinte din textul de evaluare să nu existe în vocabularul sistemului, deci nici în vocabularul modelului de limbă. În consecință, aceste cuvinte nu pot apărea în transcrierile rezultate. Ele sunt numite *cuvinte în afara vocabularului (out of vocabulary - OOV)* și nu pot fi prezise în modelul de limbă. Un procent de cuvinte OOV mic duce la o performanță bună a sistemului de recunoaștere a vorbirii.

Revenind la evaluarea completă a sistemului de RAV, trebuie menționat unul dintre criteriile de performanță, anume *rata de eroarela nivel de propoziție (sentence error rate - SER).* Aceasta se calculează ca fiind raportul între numărul de propoziții eronate, care au cel puțin o greșeală, și numărul total de propoziții din transcrierea de referință:

$$SER[\%] = \frac{Nr.\ propoziții\ eronate}{Nr.propoziții\ \text{în}\ transcrierea\ de\ referință} * 100 \quad (2.3.1)$$

Acest criteriu devine util doar pentru aplicațiile de recunoaștere vocală în care apariția unei erori devine fatală (ex. Aplicații de securitate).

Cel mai relevant și important criteriu de performanță este *rata de eroare a cuvintelor (word error rate - WER).* Acesta se referă la cât de mult cuvântul returnat diferă de transcrierea de referință. Pentru calculul acestei rate, trebuie calculată, în primul rând, distanța minima între cuvântul rezultat și cel correct, obținând astfel numărul minim de cuvinte substituite, inserate, respective șterse. După aliniere, WER se calculează pe baza următoarei formule:

$$WER[\%] = \frac{Nr.\ erori\ de\ inserție\ Nr.erori\ de\ substituție\ Nr.erori\ de\ ștergere}{Nr.cuvinte\ \text{în}\ transcrierea\ de\ referință} * 100 \quad (2.3.2)$$

Deși acest criteriu este cel mai des utilizat, acesta nu tine cont de faptul că o eroare de subtituție ar putea fi reparată dacă s-ar reporta la procentele de caractere eronate. De aceea, s-a introdus un al treilea criteriu, cel care calculează *rata de eroare la nivel de character (character error rate - ChER)* . Procedeul de calcul este extreme de similar cu cel al ratei de eroare a cuvintelor, singura diferență fiind raportarea făcută, mai exact caracterul.

$$ChER[\%] = \frac{Nr.\ erori\ de\ inserție\ Nr.erori\ de\ substituție\ Nr.erori\ de\ ștergere}{Nr.caractere\ în\ transcrierea\ de\ referință} * 100 \quad (2.3.3)$$

În figura următoare sunt ilustrate componentele necesare procesului de evaluare a unui sistem de recunoaștere a vorbirii:

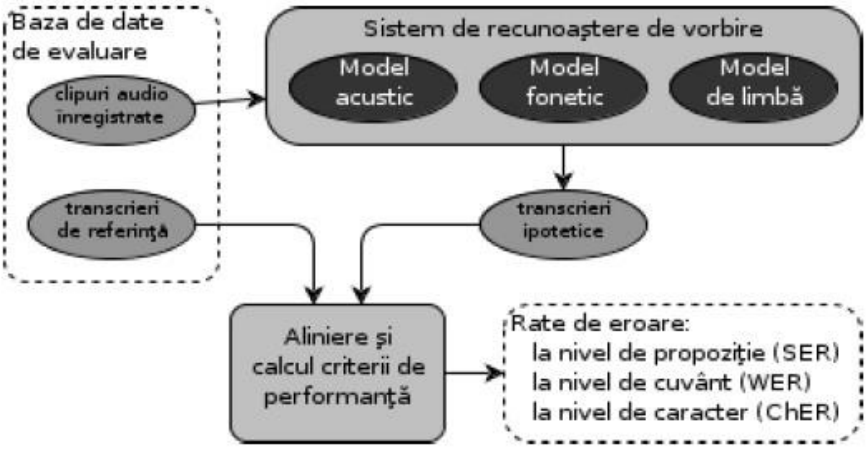


Figura 2.3.1 Evaluarea completă a sistemului RAV [10]

CAPITOLUL 3

ARHITECTURA DE COMUNICAȚIE CLIENT – SERVER

Modelul Client–Server a fost formulat pentru prima dată în anii 1960-1970 de către cercetătorii firmei Xerox în contextul design-ului unui limbaj de programare pentru rețelele de calculatoare (*Decode-Encode - DEL*). La vremea aceea acest model era utilizat doar pentru a conecta mainframe-urile[5]. Cu trecerea anilor, calculatoarele au evoluat, înlocuind acele terminale, însă conceptele modelului Client-Server au rămas neschimbate. Acesta se referă la o arhitectură software constituită din două părți, sistemul client și cel server. Cele două componente interacționează și formează o rețea [13]. În prima parte a acestui capitol vor fi prezentate noțiunile teoretice ce vizează acest model.

Motivele principale ce determină pentru folosirea intensă a acestui model sunt:

- stocarea informaţiei într-un singur loc, de unde poate fi redistribuită cu uşurinţă către clienţi;
- dedicarea resurselor de calcul (a serverelor) unor sarcini specifice cum ar fi spre exemplu poşta electronică – unde este implicată mutarea informaţiilor în siguranţă dintr-un punct în altul.

În vederea implementării practice a Sistemului Complet de Automatizare, ce reprezintă scopul acestei teze, autorul s-a bazat pe un asemenea model Client-Server. Aplicația software care folosește sistemul de recunoaștere automată a vorbirii, mai exact face transcrierea fișierului audio într-un text, rulează pe unul din serverele laboratorului SpeeD. Cum s-a demonstrat deja robotul NAO suportă conexiune la Internet. Pentru a putea exemplifica recunoașterea comenzilor vocale, robotul trebuie să comunice cu aplicația ce oferă servicii de tipul Audio-Text *(Speech-to-Text – S2T)*. Plecând de la premisele prezentate, a fost necesară alegerea unei arhitecturi de comunicație între cele două componente principale, robotul NAO și aplicația server ce rulează pe mașinile SpeeD. Marele avantaj al soluției alese de autorul acestei lucrări, este faptul ca robotul NAO comunică punct-la-punct prin Internet cu serverul SpeeD, nefiind nevoie de aplicații intermediare, cum este cazul arhitecturii deja existente.

În capitolul curent, după descrierea teoretică, va fi prezentată schema soluției propuse, protocolul utilizat de aplicația de transcriere, dar mai ales pașii urmați astfel încât soluția să devină funcțională. Protocolul de comunicație folosit de aplicația de RAV cât și aplicația software în sine au impus anumite restricții ce au dus la efectuarea unor setări speciale ale robotului.

[5] *Mainframe*-urile în prezent sunt computere cu o putere de calcul foarte mare, însă în contextul actual se referă la primele calculatoare dezvoltate de IBM.

3.1 GENERALITĂȚI

Modelul Client – Server este o structură sau arhitectură ce acționează ca o aplicație distribuită care partajează procesarea între furnizorii de servicii numiți servere și elementele care solicită servicii, numite clienți. Clienții și serverele comunică printr-o rețea de calculatoare, de obicei prin Internet, având suporturi hardware diferite, dar pot rula și pe același sistem fizic [13].

Arhitectura Client-Server este un tip de arhitectură stratificată, împărțind o aplicație în trei componente de bază:

- Clientul;
- Infrastructura rețelei;
- Serverul.

De asemenea, trebuie menționat faptul că aproape toate structura Internetului se bazează pe acest model. Multe milioane de servere, ce rulează continuu, sunt conectate la rețeaua globală.

Marea majoritate a serviciilor de Internet au următoarele componente:

- un program/aplicație software în rol de client;
- un program/aplicație software în rol de server;
- conexiuni între client și server;
- conexiuni între mai multe servere;
- conexiuni directe între clienți.

Un server este un program de aplicație care furnizează servicii altor aplicații (numite aplicații client), aflate pe același calculator sau pe calculatoare diferite. De obicei, aplicația server așteaptă conexiuni din partea aplicațiilor client. Se mai numește server și calculatorul de mare putere, pe care rulează una sau mai multe asemenea aplicații. Acesta operează continuu în rețeaua sa și așteaptă solicitări din partea altor calculatoare din rețea. Serverele pot fi folosite simultan și pentru alte scopuri, dar când nevoile o cer, ele pot fi rezervate exclusiv pentru funcția de server. Cuvântul *server* provine din cuvântul englezesc *to serve* – a servi: calculatorul server poate în principiu deservi întreaga rețea de calculatoare clienți, pentru a asigura accesul la toată paleta de forme de conectare și servicii. Deseori unul și același computer poate juca ambele roluri, și de server, și de client, în același timp. Numele de *server* este un alt termen pentru *Host computer* – computer gazdă, spre deosebire de alte elemente "inteligente" din rețea cum ar fi routerele și switch-urile. Serverele deservesc resurse hardware care sunt partajate și pot uneori fi comandate de către calculatoarele-client, cum ar fi imprimante (atunci serverul se numește *print server*) sau sisteme de fișiere (atunci el se numește *file server*) [14]. Această partajare permite un acces și o securitate mai bună; ea poate reduce cheltuielile pentru dispozitivele periferice.

Un client este o componentă hardware sau software care accesează un serviciu pus la dispoziție de către un server. Serverul este de multe ori (dar nu întotdeauna) pe un alt sistem informatic, caz în care clientul accesează serviciul prin intermediul unei rețele. Termenul a fost aplicat pentru prima dată dispozitivelor care nu erau capabile sa ruleze propriile programe, dar puteau interacționa cu computerele de la distanță printr-o rețea.

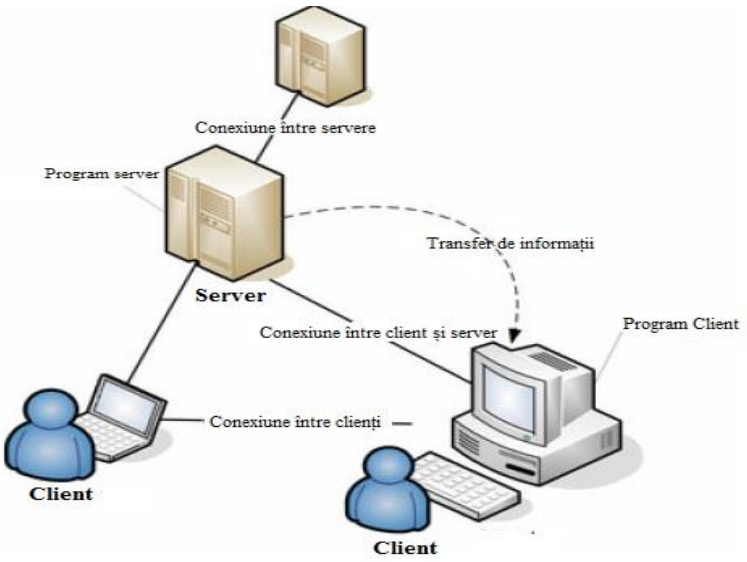


Figura 3.1.1 Arhitectura Client-Server[15]

În figura 3.1.1 este reprezentată o arhitectură tipică Client-Server. Clienții sunt figurați prin PC-uri, iar aplicațiile server prin simbolul standard al unui dispozitiv hardware de server. De asemenea, sunt specificate și conexiunile directe între diverși clienți, respectiv servere. Procesul de transfer al fișierelor între un client și server se numește descărcare (download) iar opusul său încărcare (upload).

3.2 SOLUȚIA PROPUSĂ – COMUNICAȚIA ROBOT SERVER

Cum s-a specificat deja, soluția aleasă de autor este de a-l face pe robotul NAO să comunice direct cu serverul pe care rulează aplicația RAV, printr-o conexiune de tipul punct-la-punct, adică fără aplicații intermediare care să facă tranziția datelor. O rețea punct-la-punct (*peer-to-peer - P2P*) este o rețea în care nodurile interconectate („partenerii”) distribuie resurse între ei fără a folosi un sistem de administrare centrală. Așadar, în figura de mai jos este ilustrată schema soluției.

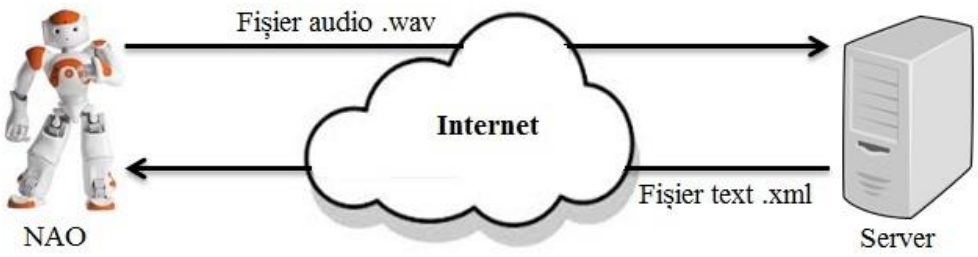


Figura 3.2.1 Schema soluției de comunicație

3.2.2 PROTOCOLUL UTILIZAT DE APLICAȚIA RAV

În capitolul anterior s-au prezentat în detaliu conceptele teoretice ce stau la baza unui sistem de recunoaștere automată a vorbirii. O aplicație a acestui sistem o reprezintă soluția software de transcreiere Speech-to-Text. Autorul acestei teze utilizează această aplicația *LiveTranscriberText*, mai precis din pachetul Java *org.etti.speech.transcriber.client* clasa *DummyTranscriberClient.java*, în vederea obținerii transcrierii comenzilor vocale în text. Ca pas premergător al alegerii soluției optime de comunicație au trebuit înțelese regulile impuse de implementarea software, în limbajul Java, a aplicației. După un număr de rulări succesive ale programului și după parcurgerea acestuia s-au constatat următoarele două aspecte importante:

- Aplicația de transcriere audio-text utilizează protocolul de comunicație TCP/IP, mai exact conexiune prin socluri. Acest tip de conexiune reprezintă un mecanism de intercomunicare între procese. Procesele care comunică între ele pot fi pe același calculator sau pe calculatoare diferite. Comunicarea cu ajutorul soclurilor se bazează pe modelul client-server. Procesul server crează un soclu și numele acestui soclu va fi cunsocut de către procesele client. Înaintea comunicării, procesul client crează un soclu anonim și cere conectarea la soclul cu numele respectiv. Daca s-a reușit conectarea, atunci se retunează câte un descriptor de fișier pentru procesul client și procesul server. Comunicarea cu ajutorul soclurilor este bidirecțională.

- Aplicația de transcriere audio-text este realizată pe modelul Client-Server și folosește un protocol bazat pe mesaje de tip xml. XML (*eXtensible Markup Language*) este un meta-limbaj utilizat în activitatea de marcare structurală a documentelor, fiind bazat pe anumite convenții. Un document XML conține două părți principale: prologul și elementul rădăcină. Prologul conține primele două linii ale unui mesaj XML, prima dintre ele reprezentând declarația XML (opțională) ce conține informații despre versiunea utilizată și tipul codării pentru text, iar cea de-a doua linie este un comentariu. Elementul rădăcină corespunde unui element unic, care poate deveni părinte pentru altele. Acesta indică structura logică a documentului, conținând informațiile corespunzătoare. Un element tip cuprinde o etichetă de deschidere, conținutul său și o etichetă de închidere. Conținutul elementului poate fi constituit din date de tip caracter, alte elemente imbricate sau combinații între cele două. Elementele pot avea atribute, ce pot fi inserate după eticheta de deschidere, ele reprezentând o alternativă la inserarea informațiilor într-un document. Utilizarea atributelor permite obținerea unui mai mare grad de felxibilitate în procesul prelucrărilor [16].

Așadar, în vederea elaborării soluției propuse pentru comunicație robot NAO și serverul ce susține aplicația de transcriere audio-text s-a ținut cont atât de protocolul de comunicație, cât și de setul de reguli impus de tehnologia XML. În figura 3.2.2 este reprezentată schema schimbului de mesaje între cele două entități participante la comunicație. În cele ce urmează vor fi descrise mesajele ilustrate.

Figura 3.2.2 Diagrama schimbului de mesaje XML

S-a menționat că aplicația S2T funcționează pe protocolul TCP/IP. Privind stiva acestuia, avem 4 nivele: Nivel Acces la rețea, Nivel Rețea/Internet, Nivel Transport, Nivel Aplicație.

- *Nivel de Aplicație* – la acest nivel are loc schimbul mesajelor XML.

Acest nivel include protocoalele de comunicație utilizate de cele mai multe aplicații pentru furnizarea de servicii utilizatorilor sau pentru schimbul de date pe conexiuni de rețea stabilite de protocoalele de nivel inferior. În majoritatea implementărilor, nivelul aplicație tratează nivelurile inferioare ca o "cutie neagră" care oferă o infrastructură sigură de comunicații, deși majoritatea aplicațiilor cunosc adresa IP sau portul folosit. Majoritatea protocoalelor de la nivelul aplicație sunt asociate cu modelul client-server. Serverele au de obicei asociate porturi fixe, atribuite de IANA, în schimb, clienții folosesc porturi temporare.

Aplicația de recunoaștere vocală folosește în acest strat un protocol care poate fi rezumat ca un protocol cerere-răspuns între client și server. La inițierea unei conexiui cu serverul, clientul trebuie să furnizeze o adresă de soclu (*eng.* socket) compusă din adresa IP și un număr de port. După stabilirea conexinuii cu succes, clientul trebuie să se autentifice folosind un nume de utilizator și o parolă. Acestea sunt transmise prin mesajul xml *authentificationRequest*. După verificarea credențialelor, serverul poate răspunde, în mesajul *authentificationResponse* cu unul din cele trei variante posibile de răspuns. Credențialele folosite de autorul acestei teze sunt cele puse la dispoziție de laboratorul SpeeD.

```
CLIENT:
<authentificationRequest
password="d1b0efbfe8bba3e60ad5952d02d52a2c6ef39da8e60eb068b5ee3f6b2dbc2aae
username="ARTIE"/>
```
- către server se transmite un șir de caractere obținut în urma criptării parolei prin intermediul unui algoritm de criptare de tip SHA256 pe 32 de biți;

```
SERVER:
<authentificationResponse result="OK"/>
```
- autentificarea a avut succes

```
<authentificationResponse result="Failed"/>
```
- autentificarea a eșuat

Dacă serverul nu dă nici un răspuns într-un interval de timp prestabilit clientul emite următorul mesaj:

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

Dacă autentificarea a avut succes, clientul va interoga configurațiile pe care serverul le suportă prin mesajul *getSupportedConfigurationsRequest*, iar serverul va trimite un mesaj cu toate serviciile pe care acesta le poate oferi *getSupportedConfigurationsResponse*.

În particular, pentru aplicația în discuție configurațiile suportate de server sunt:

- limba pentru care se face transcrierea audio în text – în cazul de față limba română;
- identificatorul domeniului, adică baza de date de referință utilizată în transcriere – pentru exemplificarea practică a acestei teze s-a implementat un domniu particular ce inlcude comenzile ce vor fi date robotului, identificatorul acestuia fiind 14;
- informații referitoare la tipul de codare și la frecvența fișierului audio care va fi trimis serverului spre prelucrare;

```
CLIENT:
<getSupportedConfigurationsRequest/>
SERVER:
<getSupportedConfigurationsResponse>
    <languages>
        <language name="Romanian">
      <domain examplePhrases="1. mergi un metru 2. mișcă mâna dreaptă 3. ridică-
te în " id="14" name="NAO Commands" ratings="1:incorrect, 5:correct"/>
      </languages>
      <audioStreams audioFrequencyBands="narrow,wide" encodings="PCM_SIGNED"/>
</getSupportedConfigurationsResponse>
```

Datele conținute de fișierul audio au restricția, din partea aplicației de recunoaștere audio, de a fi codate prin modulație în cod de impulsuri (PCM – *Pulse Code Modulation*) cu semn. Această metodă, prescurtată PCM este definită în standardul ITU-T G.711 ca fiind tehnica de codare a semnalului vocal, fiind utilizată și în telefonia digitală pentru a coda semnalul de voce. Primul pas în conversia semnalului vocal analogic într-unul digital este filtrarea lui, adică limitarea la o bandă de frecvență. Următorul pas este eșantionarea, la o frecvență care să respecte teorema eșantionării, $F_e > 2 * F_m$, în cazul aplicației în discuție eșantionarea se face la 16 kHz. Filtrarea are rolul de a preveni apariția fenomenului de aliere. După ce a avut loc eșantionarea, următorul pas este compresia semnalului, proces ce se realizează prin cunatificare (regulată - același număr de biți pentru fiecare probă sau neregulată – bazat pe legile de compresie A și μ, cu scopul de a reduce zgomotul de cuantificare).

În aplicația de transcriere audio-text, datele care trebuie trimise la server trebuie stocate într-un fișier de tipul .WAV, ceea ce înseamnă ca au fost codate pe 16 biți PCM cu semn. Principalul avantaj al acestui format .WAV e dat de faptul că nu este efectuată nicio comprimare pe fluxul audio cuantificarea fiind regulată (16 biți/eșantion) și nicio informatițe audio nu este pierdută.

În continuare, clientul are nevoie de un număr de port pentru a ști unde să trimită datele, face această solicitare prin *getAudioDataPortRequest*, iar serverul în mesajul de răspuns îi transmite numărul de port alocat, mai exact 50001.

```
SERVER:

<getAudioDataPortResponse port="50001"/>
```

Odată ce clientul a primit numărul de port, inițiază o cerere de transcriere cu toate configurațiile de care are nevoie și începe transmiterea semnalului acustic sub forma unui flux de octeți. Transmiterea datelor este transparentă din punctual de vedere al mesajelor XML.

```
CLIENT:

<getTranscriptionRequest>
      <domain id="14"/>
      <audioStream audioFrequencyBand="wide" encoding="PCM_SIGNED"/>
      <transcriptionOptions     phraseAlternatives="yes"     processedText="yes"
rawText="yes" speakerInfo="yes" timing="yes"/>
</getTranscriptionRequest>
```

Serverul anunță acceptul pentru începerea primirii datelor. Datele sunt salvate în memorie, prelucrate conform teoriei prezentate în capitolul 2, apoi transmise către client.

```
SERVER:


<startTranscriptionAck/>
<getTranscriptionResponse bestProcessedText="stai jos" bestRawText="stai jos"
duration="3900" logConfidenceScore="0.0" newParagraph="true" startTime="0">
```

Pe lângă transcrierea semnalului audio, în mesajul de răspuns serverul oferă și alte informații legate de prelucrarea efectuată, cum ar fi: durata, cea mai bună variantă de transcriere, respectiv cea mai proastă și date despre vorbitor. Pentru a marca încheierea transcrierii serverul trimite un mesaj *doneTranscriptionAck*.

```
SERVER:


<doneTranscriptionAck/>
```

- *Nivel de Transport* – se regăsește imediat sub nivelul de Aplicație. Acesta se ocupă cu probleme legate de siguranță, control al fluxului și corecției de erori. El este proiectat astfel încât să permită conversații între entitățile pereche din gazdele sursă, respectiv, destinație. În acest sens au fost definite două protocoale capăt-la-capăt. Primul din ele, TCP (*Trasmission Control Protocol*) un protocol sigur, orientat pe conexiune care permite ca un flux de octeți trimiși de pe o mașină să ajungă fără erori pe orice altă mașină din inter-rețea. Acest protocol fragmentează fluxul de octeți în mesaje discrete și pasează fiecare mesaj nivelului internet. TCP tratează totodată controlul fluxului pentru a se asigura că un emițător rapid nu inundă un receptor lent cu mai multe mesaje decât poate acesta să prelucreze. Al doilea protocol din acest nivel, UDP (*User Datagram Protocol*), este un protocol nesigur, fără conexiuni, destinat aplicațiilor care doresc să utilizeze propria lor secvențiere și control al fluxului. Protocolul UDP este de asemenea mult folosit pentru interogări rapide întrebare-răspuns, client-server și pentru aplicații în care comunicarea promptă este mai importantă decât comunicarea cu acuratețe, așa cum sunt aplicațiile de transmisie a vorbirii și a imaginilor video.

  Protocolul utilizat pentru conexiune în cazul aplicației de recunoaștere vocală este protocolul TCP; acesta segmentează mesajele de date primite de la nivelul Aplicație în părți mai mici, numite segmente. Principalul avantaj al TCP (în comparație cu UDP) este faptul că mesajele primite sunt reasamblate la destinație în aceeași ordine în care au fost trimise de la sursă și TCP garantează că datele au fost primite, ceea ce este vital în cazul transmiterii unui mesaj audio.

- *Nivel de Rețea (Internet)* – se află sub nicelul de Transport. Scopul inițial al nivelului Rețea (*Internet Protocol*) era să asigure rutarea pachetelor în interiorul unei singure rețele. Odată cu apariția interconexiunii între rețele, acestui nivel i-au fost adăugate funcționalități de comunicare între o rețea sursă și o rețea destinație.În stiva TCP/IP, protocolul IP asigură rutarea pachetelor de la

o adresă sursă la o adresă destinație, folosind și unele protocoale adiționale. Determinarea drumului optim între cele două rețele se face la acest nivel.

Comunicarea la nivelul IP este nesigură, sarcina de corecție a erorilor fiind plasată la nivelurile superioare (de exemplu prin protocolul TCP). În IPv4 integritatea pachetelor este asigurată de sume de control.

- *Nivel Acces la Rețea* – primul strat din stiva TCP/IP. Acesta se ocupă cu toate problemele legate de transmiterea efectivă a unui pachet IP pe o legătură fizică, incluzând și aspectele legate de tehnologii și de medii de transmisie.

3.2.2 ROBOTUL NAO – CLIENT

Scopul acestei teze este de a face robotul NAO să răspundă la comenzi vocale în limba română. Pentru a atinge acest scop este necesar ca robotul să comunice cu partea de server a aplicației de transcriere audio-text. Până în prezent, această legătura se putea face doar daca robotul și serverul pe care rulează aplicația erau în aceeași rețea locală (*LAN – Local Area Networking*) sau în rețele diferite, prin utilizarea unei aplicații intermediare pe post de pod între cele două. Această aplicație intermediară prelua fișierul audio prin protocol de email și îl trimitea mai departe serverului. Dezavantajul acestei metode este dat de timpul de întârziere introdus. Din considerente necunoscute la momentul implementării soluției mai sus prezentate, robotul NAO nu putea comunica mai departe de rețeaua sa locală.

Așadar, pentru a putea fi implementată soluția propusă, mai exact a-l face pe robotul NAO client în cadrul aplicației de transcriere audio-text (S2T), autorul acestei lucrări a parcurs următoarele etape:

- **Conexiunea prin SSH**

Pentru a putea a avea acces amănunțit la resursele robotului este necesară conectarea, de la distanță, prin protocol SSH menționat în primul capitol, și nu prin utilitarul Coregraphe.

SSH este un protocol care permite crearea unei sesiuni de lucru la distanță, transferul de fișiere și crearea unor canale de comunicație pentru alte aplicații, toata transmisia fiind siguă împotriva atacurilor intrușilor. Securitatea și confidențialitatea transmisiei este asigurata prin criptare. Integritatea se bazează pe trimiterea unor sume de control criptografice. Autentificarea serverului se face prin criptografie asimetrică, serverul având o cheie secretă, iar clienții dispun de cheia publică corespunzatoare. Autentificarea clientului se face fie prin criptografie asimetrică, ca și în cazul autentificarii serverului (dar bineînteles folosind altă pereche de chei), fie cu parola clasică, dată de client dupa autentificarea serverului [17].

În cazul acestei lucrări, clientul care inițiază o conexiune prin SSH este calculatorul utilizatorului, iar serverul este robotul NAO. Pentru a suporta asemenea conexiune, robotul trebuie să aibă instalat pe sistemul său de operare un server SSH. Cum s-a specificat și în capitolul 1, aproape orice SO bazat pe Linux are implicit instalat un asemenea server.

Stabilirea conexiunii poate fi descrisă prin următoarele etape:

1. Clientul și serverul se înțeleg asupra unei chei de sesiune, în continuare, întreaga comunicație fiind criptată cu această cheie de sesiune.
2. Clientul autentifică serverul. În acest scop, serverul trimite rezultatul semnării cheii de sesiune cu cheia sa secretă. Clientul verifică semnătura folosind în acest scop cheia publică a serverului.
3. Serverul autentifică la rândul său clientul. În funcție de configurația serverului, poate accepta autentificare cu criptografie asimetrică, folosind același protocol (dar bineînțeles

fară posibilitatea trimiterii de către client a cheii sale publice), sau poate cere clientului o parola.

SSH nu permite numai sesiuni de lucru prin rețea, ci și alte aplicații. Astfel, o data deschis un canal securizat, pachetele vehiculate pot fi destinate mai multor aplicații, lista celor mai importante fiind:

- Sesiune de lucru (în mod text);
- Transfer de fișiere (cunoscut ca sftp sau scp);
- Retransmiterea porturilor TCP;
- Retransmiterea unui server X (clientul SSH acționează ca server X și retransmite cererile de la clienți catre serverul SSH);

În etapa inițială de implementare practică, autorul acestei teze a făcut înregistrari audio cu microfoanele robotului NAO pentru a testa calitatea acestora. Acestea se stochează în memoria robotului. Pentru a putea fi introduse în aplicația de recunoaștere vocală au trebuit transferate în memoria calculatorului de pe care rula la momentul respectiv aplicația S2T. Pentru a putea face acest transfer de fișiere autorul a folosit utilitarul WinSCP ce funcționează pe baza protocolului SSH.

Autentificarea clientului SSH se poate face prin parolă sau prin criptografie. Se lansează o aplicație numită *agent de autentificare* care citește cheia secretă criptată, cere cheia de decriptare pe care o memorează în RAM. La lansarea unui client SSH, acesta încearcă să contacteze agentul de autentificare - dacă agentul ruleaza în acel moment. Comunicația se face local prin primitive sigure oferite de sistemul de operare al mașinii client. Clientul SSH trimite agentului cheia de sesiune, iar agentul îi returnează semnătura.

S-a utilizat programul gratuit PuTTY pentru a emula un terminal fiind un client pentru SSH și nu numai. La deschiderea acestuia trebuie furnizate câteva date necesare deschiderii conexiunii, cum ar fi numele de host sau adresa IP (adresa IP robotului NAO), portul utilizat și tipul conexiunii (SSH). În figura de mai jos este ilustrată fereastra principală a utilitarului PuTTY.

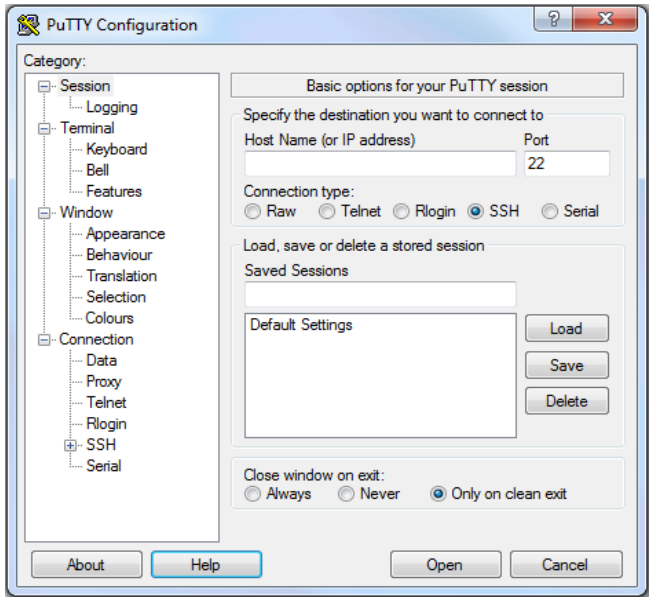


Figura 3.2.3 Utilitarul PuTTY

La încărcarea sesiunii (Load) se deschide un terminal în linie de comandă ce permite autentificarea pe sistemul de operare al robotului. Limbajul utilizat este cel specific sistemelor bazate pe Linux, iar comenzile utile pentru depanarea din punctul de vedere al rețelelor de calculatoare sunt cele studiate la materiile de specialitate. Pentru a putea efectua modificări

importante la configurașia sistemului de operare, un utilizator trebuie să se autentifice cu drepturi de "rădacină", root.

- **Deblocarea Firewall**

Una din contribuțiile autorului acestei teze constă în găsirea unei soluții astfel încât robotul NAO să fie capabil a trimite pachete în Internet, indiferent de destinație.

În fază premergătoare s-a constatat că NAO poate primi notificări și se pot descărca aplicații direct de pe site-ul fimei Aldebaran pe sistemul său de operare. Cu toate acestea, la încercarea trimiterii unor pachete de tipul ecou-răspuns (prin intermediul instrumentului de rețea *ping*) către adrese IP externe rețelei locale, procentajul de succes era nul.

S-a constatat că restricțiile erau impuse de firma producătoare Aldebaran cu scop de securitate, mai exact protecția de tip Firewall era activată.În urma unor cercetări pe site-urile de specialitate s-a gasit o modalitate de rezolvare a acestei problem, ce implică modificarea unor setări software ale robotului. În rețelele de calculatoare, un firewall, denumit și paravan de protecție este un dispozitiv sau o serie de dispozitive configurate în așa fel încât să filtreze, să cripteze sau să intermedieze traficul între diferite domenii de securitate pe baza unor reguli predefinite [18].

Consultând documentația pusă la dispoziție, s-a constatat că exsită mai multe moduri de lucru cu NAOqi din punct de vedere al securității. Cel util pentru scopul autorului este cel numit "dezvoltare" (development). În figura următoare este ilustrat modul curent activat.

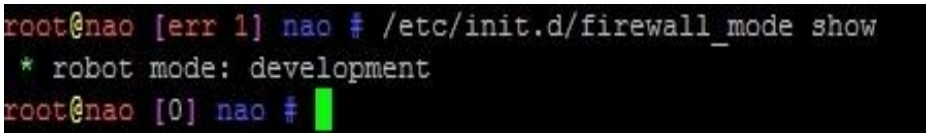


Figura 3.2.4 Deblocare Firewall-ului

În urma efectuării acestei setări, la încercarea rulării "ping" către orice adresă IP din Internet, rata de succes a transmisiunii a fost 100%.

- **Instalare Java**

Aplicația sistemului de recunoaștere automată a vorbirii, atât partea ce rulează pe server cât și cea de client, a fost dezvoltată de echipa laboratorului SpeeD în limbajul de programare Java. Prin urmare robotul NAO trebuie să fie capabil a rula programe scrise în acest limbaj de programare arhi-cunoscut.

Documentația software a robotului NAO descrie pașii ce trebuie urmați de utilizator pentru a putea executa programe Java.

În primul rând trebuie descărcat de pe site-ul celor de la Oracle, așa numitul JRE pe 32 de biți, compatibil cu Linux. JRE, cunoscut ca și Java Runtime, face parte din kitul de dezvoltare Java (JDK), un set de utilitare de programare pentru dezvoltarea aplicațiilor în Java. JRE oferă un minim de cerințe necesare execuției unei aplicații Java. Acesta constă în mașina virtuală implicită JVM, clasele de bază și fișierele de suport. Progamul se decarcă sub forma unei arhive pentru Linux cu extensia .tar.gz.

Cu ajutorul programului de transfer de fișiere, mai sus menționat, WinSCP, arhiva JRE este adusă în memoria robotului. Trebuie extrase fișierele din această arhivă. Pentru această etapă se folosește o comandă specifică. Extracția se face într-un fișier nou creat pe robot, sub denumirea java. Comenzile utilizate sunt următoarele:

```
tar xvfz jre-7u60-linux-i586.tar.gz
mkdir /home/nao/java
mv jre1.7.0_60 /home/java
// Din manualul utilitarului tar:
```

```
x - extract - extracție
v - verbose - verbozitatea
f - file    - fișierul
z - gzip    - tipul arhivei
```

Următorul pas constă în crearea și editarea unui fișier denumit “sourceme” care trebuie să conțină căile către JRE, respectiv librăriile necesare execuției unui program Java.

```
nano /home/nao/java/sourceme

export JAVA_HOME=/home/nao/java/jre
export PATH=$PATH:/home/nao/java/jre/bin
```

Pasul final este acela de a aduce pe robot proiectul Java, mai exact aplicația *LiveTranscriberClient*, prin intermediul căreia robotul se va conecta la server-ul pe care este implementat sistemul de recunoaștere automată a vorbirii. Proiectul trebuie mai întâi să fie compilat, apoi rulat. Comenzile pentru aceste două ultime etape sunt următoarele:

```
# Compilarea aplicației
javac -cp /calea/către/proiect.jar Numele_Clasei.java
# Rularea aplicației
java -cp /calea/către/proiect.jar:. Numele_Clasei
```

CAPITOLUL 4

ACHIZIȚIE, COMANDĂ ȘI CONTROL

Scopul principal al acestei teze este de a contura un Sistem Complet de Automatizare, prin intermediul unui sistem de recunoaștere vocală pentru limba română și a unui robot umanoid. Automatizarea constă în faptul că robotul este capabil să ruleze implicit comportamentul definit de autor, astfel încât la apariția unui eveniment să facă înregistrarea comenzii, să o trimita către serverul cu sistemul de RAV, cu ajutorul aplicației *LiveTranscriberClient*, să primească răspunsul, să-l interpreteze și să-l execute. Schema sistemului este cea prezentată în capitolul de Introducere ce conține trei blocuri principale:

- Blocul de Achiziție;
- Blocul de comandă și control;
- Elementul de execuție;

S-a urmărit ca aceste blocuri să păstreze descrierea prezentată în Anexa 1. Pe parcusrul acestui capitol vor fi detaliate atât funcționalitățile fiecarui bloc în parte, cât și implementările software corespunzătoare.

Sistemul Complet de Automatizare este demonstrat de autor prin reușita de a-l face pe robotul NAO să răspundă comenzilor vocale în limba română prin execuția lor. NAO are deja implementat sistem de recunoaștere automată a vorbirii, însă doar pentru limba engleză. De asemenea, el este apt să și răspundă tot în engleză cu o pronunție corectă și inteligibilă. Pentru limba română, nu suportă asemenea caracteristici, de aceea una din provocările acestei teze a fost folosirea sistemului de recunoaștere automată a vorbirii pentru limba română pe robotul NAO. Contribuția autorului, în acest stadiu final a constat în stabilirea listei de comenzi și descrierea, respectiv implementarea software a comportamentului robotului.

Autorul a elaborat o listă de comenzi simple, în care unele din ele implică mișcări de bază ale robotului NAO, altele modificările unor setări vizibile. Lista a fost concepută astfel încât să conțină cuvinte cât mai diferite posibil astfel încât sistemul de recunoaștere automată a vorbirii să nu introducă erori la transcriere. Lista comenzilor este următoarea:

- “Ridică-te în picioare”;
- “Stai jos”;
- “Mergi un metru”;
- “Mișcă mâna dreaptă”;
- “Fă-ți ochii roșii/albaștrii/verzi”;
- “Vorbește mai tare”;
- “Vorbește mai încet”;
- “Cât e ceasul?”;
- “Ce dată e azi?”;

4.1 COREGRAPHE

Mediul de lucru ales pentru definirea comportamentului robotului NAO este utilitarul pus la dispoziție de firma Aldebaran, Coregraphe. Această alegere este întărită de facilitățile pe care acest program le pune la dispoziția utilizatorului.

Coregraphe este în definitiv, o aplicație desktop pe mai multe platforme, care permite crearea animațiilor și comportamentelor, testarea lor pe un robot simulat sau direct pe unul real. De asemenea, acesta permite atât monitorizarea cât și controlul robotului NAO. Cum s-a afirmat mai sus, avantajul folosirii acestui program este dat de faptul că utilizatorul nu trebuie sa scrie aplicațiile în alte medii de dezvoltare pe care să le apeleze ulterior în linie de comandă, ci poate edita macro-blocurile existente în librariile Coregraphe, care sunt open-source, adică pot fi modificate funcție de dorințele dezvoltatorului.

Coregraphe face interacțiunea utilizator – robot mult mai ușoară prin multitudinea setărilor care le pune la dispoziție. Există funcții predefinite stocate în biblioteci, grupate funcție de modulul pe care-l folosesc, așa cum se poate observa în figura 4.1.

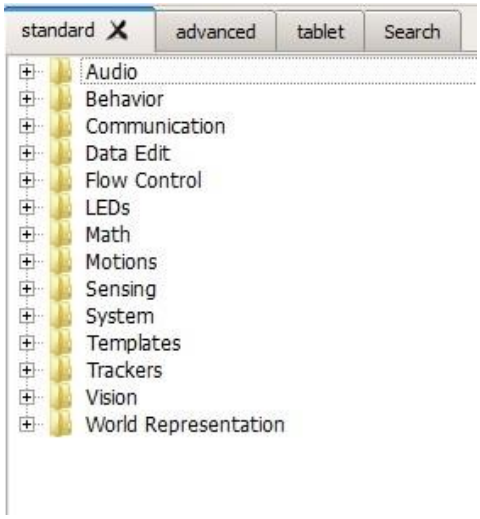


Figura 4.1.1 Fereastra de biblioteci

Limbajul de programare folosit în descrierea funcțiilor din Coregraphe este Python. Acesta este un limbaj de programare dinamic multi-paradigmă, a fost creat în 1989 de programatorul olandez Guido van Rossum. Python pune accentul pe curățenia și simplitatea codului, iar sintaxa sa le permite dezvoltatorilor să exprime unele idei programatice într-o manieră mai clară și mai concisă decât în alte limbaje de programare ca C. În ceea ce privește paradigma de programare, Python poate servi ca limbaj pentru software de tipul *object-oriented*, dar permite și programarea imperativă, funcțională sau procedurală. Sistemul de tipizare este dinamic iar administrarea memoriei decurge automat prin intermediul unui serviciu „gunoier” (*garbage collector*). Alt avantaj al limbajului este existența unei ample biblioteci standard de metode.

Pentru a putea folosi metodele predefinite, modulele pot fi translatate din fereastra de biblioteci în spațiul de lucru, unde vor fi reprezentate prin simboluri ca în figura 4.1.2. Unele module pot avea parametrii de intrare ce pot fi modificați de utilizator ca în cazul funcției de mișcare *Move To*. În cazul acesta, funcția *Move To* determină robotul să se deplaseze pe o singură direcție, pe o anumită distanță. Există posibilitatea setării direcției (înainte sau înapoi), unei distanțe de parcurs sub un unghi Theta ce determină o mișcare circulară. De asemenea, există opțiunea de setare a unei opriri sigure la o anumită distanță față de un obstacol.

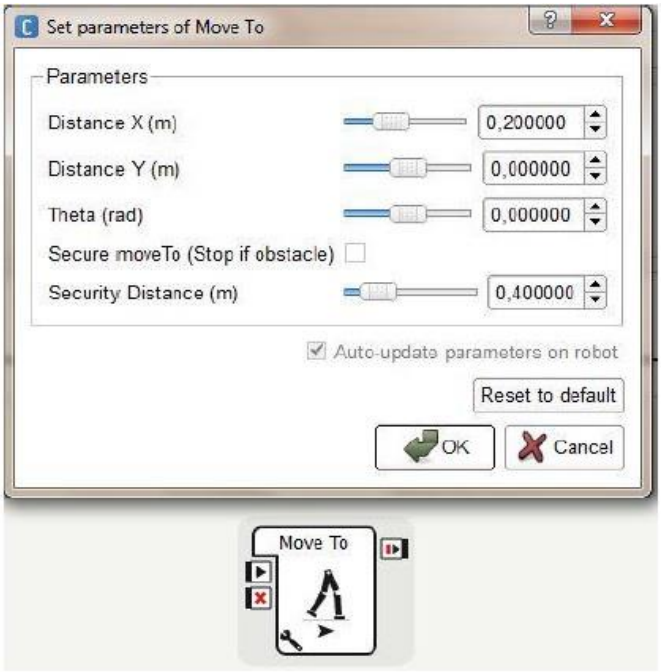


Figura 4.1.2 Macro-blocul *Move to* – parametrii și simbol

Utilizatorul are posibilitatea de a-și descrie propriile module prin intermediul editorului de Script-uri, astfel încât să poată defini comportamentul dorit pe NAO sau să le modifice pe cele predefinite. Un script este un program software sau o secvență de instrucțiuni care e interpretat sau de care se ocupă un alt program,altul decât procesorul calculatorului(ca în cazul unui program compilat). În biblioteca Coregraphe există un modul intitulat *Python Script* ce conține minimul necesar de metode (doar definirea acestora, nu și funcționalități) pentru a crea un nou script. Acesta are următoarea formă [19]:

```
class MyClass(GeneratedClass):
def __init__(self):
GeneratedClass.__init__(self)

def onLoad(self):
#put initialization code here
pass

def onUnload(self):
#put clean-up code here
pass

def onInput_onStart(self):
#self.onStopped() #activate the output of the box
pass

def onInput_onStop(self):
self.onUnload() #it is recommended to reuse the clean-up as the box is stopped
self.onStopped() #activate the output of the box
```

Clasa *MyClass* este cea care reprezintă modulul nou creat. Conform primelor 3 linii de cod, aceasta moștenește clasa *GeneratedClass,* o clasă care se generează automat la execuția unui comportament. Aceasta include toată informația necesară unui modul (ieșiri, intrări, parametrii etc.). De asemenea aceasta definește metodele implicite dintr-un script. Faptul că *MyClass* moștenește *GeneratedClass* face posibilă utilizarea acestor metode implicite. Desigur, pot fi adăugate și alte noi funcții.

Metodele corespunzătoare intrărilor modulului trebuie definite folosind sintaxa `onInput_<input-name>`, aceste metode fiind apelate de fiecare dată când intrarea este stimulată. Asemănător și în cazul metodelor corespunzătoare ieșirilor.

Există metode ce se apelează automat la încărcarea și descărcarea modulului. Metoda `onLoad` este apelată atunci când modulul este încărcat, adică atunci când se ajunge la nivelul modulului respectiv. Metoda `onUnload` este chemată la descărcarea unui modul, mai exact atunci când se iese din acesta. Este recomandat ca metoda `onUnload` să fie apelată în codul metodei corespunzătoare întrării `onStop` astfel încât modulul să se reinițializeze la oprire.

Un modul poate avea mai multe tipuri de intrări și de ieșiri. Există 4 tipuri de intrări [20]:

- *on Start* – când intrarea e stimulată, modulul începe;
- *onStop* – când intrarea e stimulată, modulul se oprește;
- *onEvent* – intrarea nu are un efect direct asupra modului, atunci când este stimulată se apelează metoda `onInput_<input_name>` și se execută codul din interiorul acesteia;
- *ALMemory Input* – nu poate fi stimulată din exterior. Este stimulată de fiecare dată când valoarea unei date stocate în ALMemory corespunzătoare acestei intrări este modificată.
- *onLoad* – este stimulată doar atunci când rularea programului ajunge la nivelul respectiv.

Există două tipuri de ieșiri:

- *onStopped* – când ieșirea este stimulată, modulul este oprit;
- *punctual* – nu are nici un effect asupra modulului, semnalul primit la ieșirea unui modul este transmis mai departe modulului părinte.

De asemenea, intrările și ieșirile mai pot fi clasificate astfel:

- *Bang* – reprezintă un eveniment simplu, nu conține date ci doar infromația care e stimulată;
- *Number* – reprezintă un eveniment ce conține date, mai exact un număr sau un vector de numere;
- *String* - reprezintă un eveniment ce conține date, mai exact un șir de caractere sau un vector de șiruri;
- *Dynamic* – reprezintă un eveniment simplu (ca și tipul Bang) sau unul ce conține date ce pot fi de orice tip .

4.2 BLOCUL DE ACHIZIȚIE

Primul bloc din structura Sistemului Complet de Automatizare este blocul de achiziție. Bibliotecile puse la dispoziție de Coregraphe conțin un grup de funcții destinate sistemului audio. Din acestea, pentru achiziționarea semnalului audio s-a folosit în primul rând macro-blocul denumit *Record*. În figura următoare este reprezentat simbolul acestei funcții, cât și blocurile ce le conține.

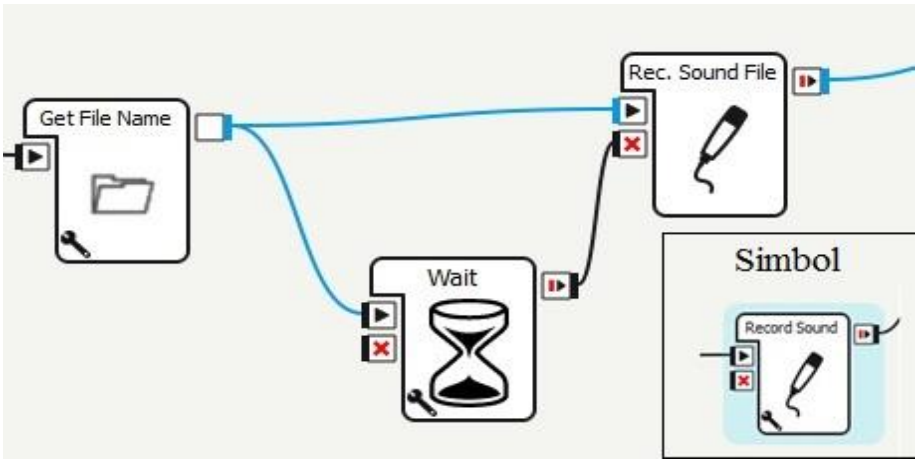


Figura 4.2.1 Sub-blocul *Record* – simbol și blocurile interne

Blocul initulat "Get File Name" este cel prin intermediul căruia se poate specifica unde în memoria robotului trebuie salvat fișierul audio înregistrat. În capitolul precedent s-a spus că aplicația *LiveTranscriberClient* a fost adusă în memoria robotului. Aceasta conține propriile directoare, iar cel de unde știe să-și ia înregistrarea pentru a o trimite spre transcriere este numit *data*. Așadar, în scriptul Get File Name a fost scrisă calea corectă unde se dorește a fi salvată înregistrarea audio.

```
/liveTranscriber/data/test
```

Ieșirea acestui bloc duce spre intrările altor două, după cum se poate observa și în figura 4.2.1. Se declanșează, astfel, începutul unei înregistrări dar și contorizarea unui interval de timp.
Blocul *Wait* conține un cronometru la expirarea căruia va fi activată ieșirea sa, care duce spre intrarea sensibilă la evenimente, de tipul *Bang,* ce determină oprirea înregistrării audio. Practic prin acest bloc se specifică durata unei înregistrări. Blocul *Rec. Sound File* este cel prin care se face înregistrarea propiu-zisă.

Pentru a alege înregistrarea optimă din punct de vedere al calității și intensității semnalului vocal, autorul acestei teze a efectuat multiple înregistrări cu diverse configurații ale robotului, variind următorii parametrii: numărul de microfoane și frecvența de eșantionare (la 16kHz și la 48kHz). Cea mai mare parte din energia semnalului vocal se regăsește între 0 și 4 kHz. Frecvența de eșantionare de 48kHz implică un spectru mai larg al semnalului, incluzând astfel componente de zgomot, fapt ce duce la o înregistrare de proastă calitate. De asemenea, cum s-a prezentat și în capitolul 2, sistemul de recunoaștere vocală recomandă ca semnalele vocale să fie eșantionate la 16 kHz, frecvență ce aduce la o calitate mai bună. Așadar, parametrii modificați în blocul "Rec. Sound File" sunt cei ce oferă o calitate satisfăcătoare a înregistrarii audio, și anume: 1 microfon (cel de pe partea dreaptă) și frecvența de eșantionare de 16 kHz.

Blocul principal de achiziție mai conține, pe lângă cel de *Record Sound* mai sus prezentat, alte două sub-blocuri, cum reiese și din figura 4.2.2.

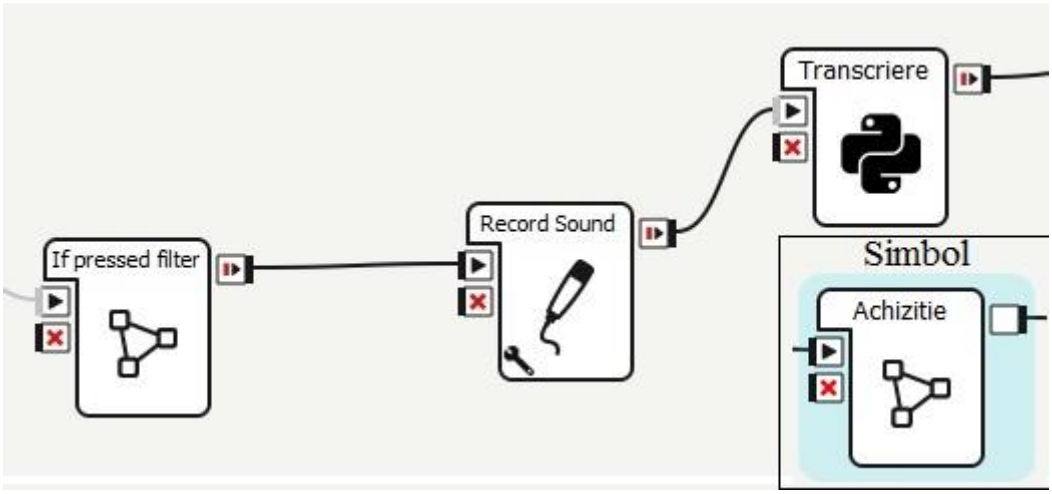


Figura 4.2.2 Blocul de Achiziție – simbol și blocurile interne

Sub-blocul *If pressed filter* a fost creat cu scopul de a emite un eveniment ce declașează următorul sub-bloc *Record Sound*, mai exact înregistrarea. La atingerea senzorului tactil central, de pe capul robotului, la detecția unei presiuni se emite un eveniment la ieșirea blocului. Scopul folosirii *If pressed filter* este de a acționa ca un declanșator pentru tot comportamentul robotului.

Sub-blocul *Transcriere* este cel prin care se face apelul aplicației *LiveTranscriberClient*. În capitolul precedent s-a prezentat schimbul de mesaje între cele două entități participante la comunicație, dar și modalitatea prin care această aplicație trebuie rulată. Prin urmare autorul a creat un nou script în cadrul căruia se face un Apel de Sistem (System-Call) către aplicația scrisă în Java.

Acest apel se face prin următoarea comandă:

```
os.system("/home/nao/java/jre1.8.0_45/bin/java -cp
/home/nao/liveTranscriber/LiveTranscriberClients.jar
org.etti.speech.transcriber.client.DummyTranscriberClient 141.85.252.230 5004
ARTIE LVCSR-ROM 14 /home/nao/liveTranscriber/data/test.wav >
/home/nao/liveTranscriber/data/std.out 2>
/home/nao/liveTranscriber/data/std_err.out")
```

`/home/nao/java/jre1.8.0_45/bin/java` – calea către executabilul Java

`-cp /home/nao/liveTranscriber/LiveTranscriberClients.jar` – calea către arhiva în care se găsește clasa ce urmează a fi rulată

`org.etti.speech.transcriber.client.DummyTranscriberClient` – clasa ce va fi rulată

Parametrii de intrare ai aplicației Client:

`141.85.252.230` - adresa IP a serverului
`5004` – numărul de port al serverului
`ARTIE` – numele de utilizator folosit pentru autentificare
`LVCSR-ROM` – parola de autentificare pe server
`14` – identificatorul domeniului din care se va face transcrierea
`/home/nao/liveTranscriber/data/test.wav` – calea și fișierul ce va fi trimis

`> /home/nao/liveTranscriber/data/std.out` – se face redirecționarea ieșirii standard (Standard Output) către un fișier

`2> /home/nao/liveTranscriber/data/std_err.out` – se face redirecționarea ieșirii de eroare (Standard Error) către un fișier

Tot în cadrul blocului de achiziție a fost necesară și o post-procesare a răspunsului primit de la server. Cum se știe deja, răspunsul serverului este sub forma unui mesaj XML, însă pentru implementarea practică este nevoie doar de textul ce conține comanda transcrisă. În acest scop, autorul a creat în aplicația *LiveTranscriberClient,* mai exact în clasa utilizată *DummyTranscriberClient,* o nouă variabilă denumită *text* în care se va scrie valoarea atributului *bestProcessedText.* Astfel, după ce serverul anunță încheierea transcrierii, în variabila *text* se va scrie comanda transcrisă din înregistrarea audio. De asemenea, autorul a creat o nouă metodă numită *writeToFile()* prin intermediul căreia scrie într-un fișier (*transcription.out*) comanda transcrisă. Această metodă este apelată la finalul metodei *main().* Cu aceste modificări, autorul are acum la dispoziție comanda nou transcrisă într-un fișier pe care-l poate citi oricând.

4.3 COMANDĂ ȘI CONTROL

Cel de-al doilea bloc principal din structura Sistemului Complet de Automatizare este blocul de Comandă și Control. Complexitatea acestuia este una ridicată datorită numărului mare de blocuri interne cât și a legaturilor dintre ele. Rolurile blocului, așa cum sugerează și numele, sunt acelea de a comanda și controla robotul. La acest nivel, textul din fișierul *transcription.out* este prelucrată astfel încât să fie identificată comanda rostită. Odată găsită comanda, se trece mai departe la controlul efectiv al robotului astfel încât să efectueze corect ce trebuie.

Implementarea propusă de autor poate fi descrisă prin figura următoare:

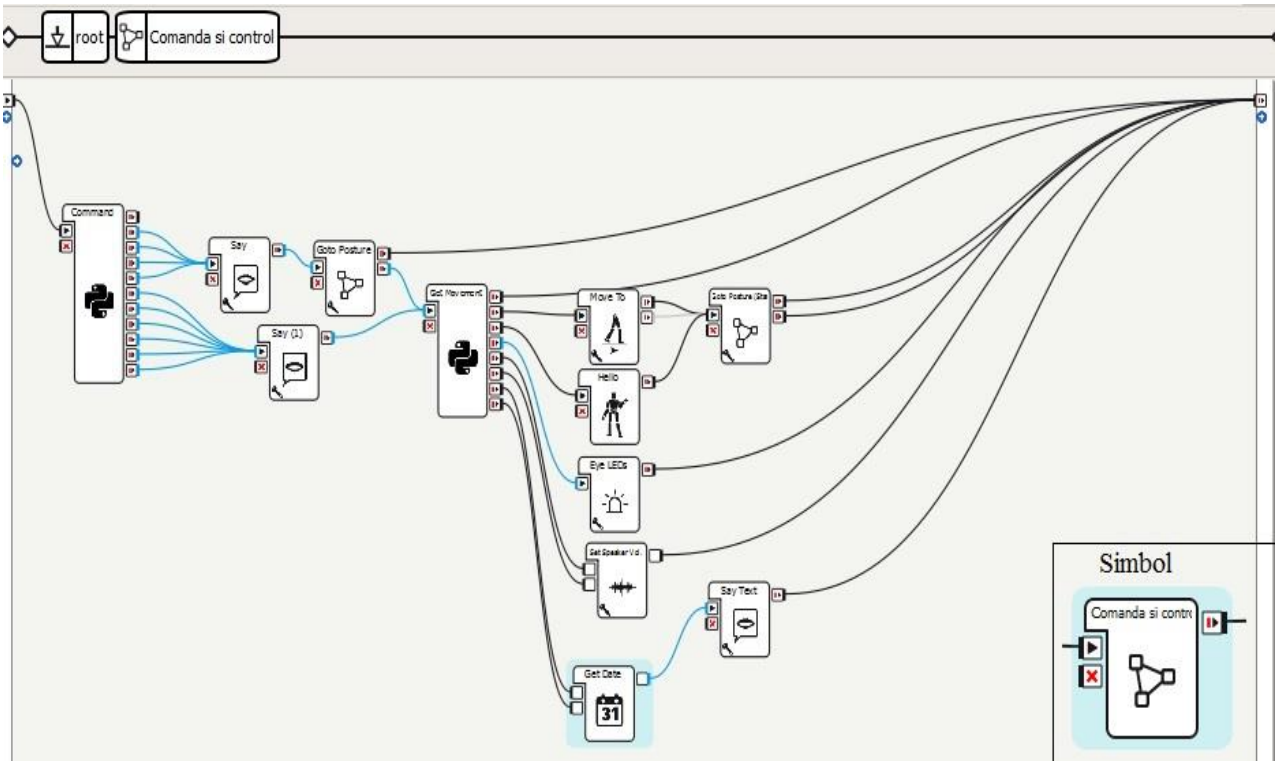


Figura 4.3.1 Blocul de Comandă și Control – simbol și blocurile interne

În continuare vor fi prezentate pe rând, fiecare din blocurile interne:

- *Comandă*

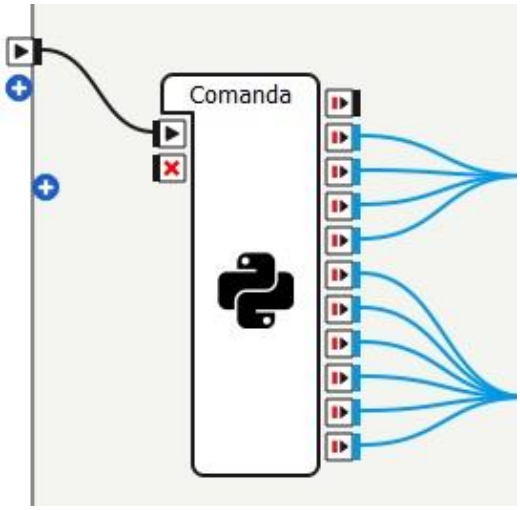


Figura 4.3.2 Sub-blocul Comandă

Figura 4.3.2 ilustrează simbolul utilizat pentru acest bloc intern și multitudinea de ieșiri ale acestuia. Pentru implementarea blocului autorul a creat un nou script în care a descris următorele funcționalități:

1. *Prelucrarea textului ce conține comanda transcrisă*

```
f = open( '/home/nao/liveTranscriber/data/transcription.out', 'r' )
self.command = f.readline()
f.close()
self.command = "".join(ch for ch in self.command if ch.isalnum())
f = open( '/home/nao/liveTranscriber/data/test.out', 'w' )
f.write( self.command )
f.close()
```

Prima coamndă deschide pentru citire fișierul *transcription.out* care se află la calea afișată. În variabila *self.command* este scris primul rând din fișierul deschis, mai exact comanda transcrisă, iar apoi se închide fișierul. Serverul oferă comanda transcrisă cu diacritice, însă deoarece ulterior variabila *self.command* va fi comparată cu șiruri de caractere, s-a ales ca din text să se elimine diacriticele și spațiile, evitându-se astfel problemele ce puteau apărea din cauza incompatibilității codurilor pentru caractere. Prin urmare, linia a 4-a din comenzile de mai sus, suprascrie variabila *self.command* astfel: la șirul gol `""` se adaugă un nou caracter, prin parcurgerea șirului din *self.command,* doar dacă acesta este de tipul alfa-numeric. Pentru o vizualizare a noii comenzi, fără diacrticie s-a creat un alt fișieri în care s-a scris textul prelucrat, adică în fișierul *test.out*.

În urma unei astfel de prelucrări, de exemplu, comanda “Ridică-te în picioare” va deveni “ridictenpicioare”.

2. *Identificarea comenzii*

Identificarea comenzii se face prin intermediul instrucțiunei condiționale de tipul *if-else*. Pentru fiecare nouă înregistrare, la acest nivel al programului, se face o comparație între variabila ce conține comanda curentă *self.command* cu prelucrările aferente și șirul de caractere care se așteaptă a fi primit. Dacă verificarea este adevărată se face apelul unei metode, ca în exemplul următor:

```
if self.command == 'staijos' :
self.jos()
elif self.command == 'ridictenpicioare' :
self.sus()
elif self.command == 'mergiunmetru' :
self.mergi()
```

```
elif self.command == 'micmnadreapt' :
self.mana()
.
.
else:
      self.error()
```

Bineînteles, asemenea comparații se fac pentru fiecare comandă din lista descrisă în introducerea capitolului curent. În cazul în care transcrierea audio-text a eșuat, iar nici una din comparațiile făcute nu este adevărată se apelează o metodă definită special pentru cazurile de eroare.

3. *Metodele corespunzătoare fiecărei comenzi*

Pentru fiecare dintre cele 9 comenzi s-a definit câte o metodă. Funcționalitățile acestora sunt similare, de aceea vor fi descrise doar câteva dintre ele. La încărcarea modulului (onLoad) au fost declarate 2 variabile globale *self.posture* și *self.text.* Funcție de comandă, în metoda respectivă *self.text* va conține traducerea comenzii în limba engleză, iar *self.posture* va conține poziția robotului în care el trebuie să ajungă pentru a executa comanda respective, doar dacă este cazul să-și schimbe poziția. În fiecare metodă se atribuie cate-o valoare corespunzătoare acestor două variabile și de asemenea, se va defini câte o ieșire din bloc prin intermediul căreia se vor pasa 3 parametrii următoarelor blocuri. Pentru a avea un feedback asupra transcrierii efectuate de sistemul de RAV, autorul tezei a ales variant în care fiecare comandă din limba romînă să aibă o traducere corectă în limba engleză, pe care robotul o va rosti. Astfel, se va ști că s-a transcris ceea ce trebuie.

O astfel de metodă are următoarea structură:

```
def sus(self):
self.text = 'Get up!'
self.posture = 'Stand'
self.onSus([self.text,self.posture,self.command])
```

Pentru comenzile care nu necesită acțiune de mișcare poziția robotului nu influențează execuția comenzii, așa că variabila *self.posture* rămâne goală, ca în cazul de mai jos:

```
def time(self):
self.text = 'Time!'
self.posture = ''
self.onTime([self.text,self.posture,self.command])
```

Cei 3 parametrii care sunt pasați următorului bloc sunt:

- *self.text* – util blocului *Say*;
- *self.posture* – util blocului *Goto Posture*;
- *self.command* – util blocului *Get Movement;*

Prin ultima linie a fiecărei metode, de exemplu *self.onTime* se activează ieșirea corespunzătoare comenzii "Cât e ceasul?". Deci pentru fiecare comandă au fost create câte o ieșire din blocul current.

- *Say* și *Say(1)*

Aceste două blocuri sunt predefinite în biblioteca Coregraphe și au ca scop rostirea unui text presetat. În cazul de față cele două blocuri au fost modificate astfel încât textul să fie primit ca parametru generat de modulul precedent *Comandă.* Totodată cele două blocuri au fost modificate pentru a putea face un transfer de parametrii de la blocul anterior la blocuri posterioare, fiind un fel de tunel. *Say* transmite la ieșire, către blocul *Goto Posture*, un vector cu două elemente de tip șir de caractere (*self.posture* și *self.command*). *Say(1)* este apelat pentru comenzile care nu implică mișcări ale robotului, de aceea el transmite la ieșire doar un parametru și anume *self.command.*

- *Goto Posture*

Blocul *Goto Posture* are drept scop să aducă robotul în poziția primită ca parametru, adică poziția din care trebuie executată comanda. Acesta este un modul predefinit care a fost modificat astfel încât să transmită și el la rândul său mai departe un parametru primit la intrare, și anume *self.command*.

- *Get Movement*

Este un bloc creat de la zero de autor cu scopul de a identifica prin intermediul comenzii ce anume trebuie să execute robotul. În esență acest modul este similar cu cel denumit *Comandă* diferența constă în faptul că acum robotul este în poziția corectă de execuție. De asemenea, o altă diferență este dată de ieșiri. În cazul modului în discuție, majoritatea ieșirilor emit evenimente ce acționează ca declanșatoare pentru modulele de execuție propiu-zise. Excepție face ieșirea *onOchi* care transmite modulului *Eye LEDs* un parametru ce are valoarea culorii corespunzătoare comenzii date "Fă-ți ochii roșii/albaștrii/verzi".

Urmărind schema blocului de Comandă și Control, au mai rămas doar modulele ce determină execuția propiu-zisă a comenzilor.

- *MoveTo*

Asupra acestui modul predefinit nu au survenit modificări. Acesta determină robotul să se deplaseze pe o singură direcție, pe o anumită distanță. Există posibilitatea setării direcției pe două axe (înainte–înapoi sau laterale) și a distanței de parcurs. De asemenea, există opțiunea de setare a unei opriri sigure la o anumită distanță față de un obstacol.

- *Hello*

Acest modul s-a ales să fie modulul ce se apelează în urma comenzii vocale "Mișcă mâna dreaptă". Instruncțiunile predefinte în acesta îl fac pe NAO să execute o mișcare animată similară unui salut, de aici și numele modulului.Ieșirile acestor două ultime module duc spre un alt bloc de tipul *Goto Posture* și nu spre ieșirea principală, deoarece s-a observat că la terminarea execuției mișcărilor robotul rămâne într-o poziție tensionată nefirească ce poate afecta anumite motoare. De aceea, autorul a decis folosirea unui alt modul care să-l aducă pe NAO în poziția de stat în picioare.

- *Eye LEDs*

Acest sub-bloc a fost definit pentru culorile folosite în lista de comenzi, culori ce sunt primite ca parametru la intrare. Controlul LED-urilor tricolore (RGB) se face cu ajutorul unei funcții predefinite ce primește ca parametru un vector al cantităților de culoare din fiecare din cele 3 culori (valoari între 0 și 255), grupul LED-urilor țintă, dar și un timp în care se va face trecerea gradată de la culoarea curentă la cea dorită.

- *Set Speaker Volume*

Scopul acestui modul este de a modifica volumul robotului, fapt ce se sesizează atunci când acesta rostește traducerile comenzilor din română în engleză. Acest sub-bloc se apelează la comenzile "Vorbește mai tare" sau "Vorbește mai încet", de aceea are nevoie de două intrări și de două metode diferite în conținutul său *onInput_onVolumeUp* și *onInput_onVolumeDown*. Volumul nu se modifică inferior sau superior la o valoare prestabilită, ci gradual cu un procentaj de 10% din valoarea maximă.

- *Get Date*

Acest modul se apelează la comenzile ce implică data sau ora curentă, de aceea și acesta are nevoie de două intrări. În interiorul acestuia sunt definite două metode. Cea care returnează timpul, *onInput_onGetTime,* a fost creată astfel încât variabila *self.string* pe care o pasează la ieșire, să conțină o concatenare de cuvinte – ora și minutul curent. Așadar acest string este dat mai departe

unui modul de tip *Say*, iar robotul răspunde cu o cursivitate logică în limba engleză “It`s hour x and y minutes”. Autorul a întâmpinat o problemă în sincronizarea ceasului de sistem a robotului cu fusul orar al României. Deși a setat hardware ceasul de sistem, la o nouă pornire a robotului acesta trecea pe fusul implicit. De aceea a fost nevoie de a aduna 3 la funcția de returnare a orei, astfel încât răspunsul dat să fie unul corect.

Metoda care returnează data, *onInput_onGetDate*, a fost și ea definită astfel încât răspunsul rostit de robot să fie unul cursiv și logic. Pentru pronunția literară a lunilor anului, autorul a definit un vocabular în care a atribut indexului fiecărei luni denumirea respectivă în limba engleză. Variabila *self.string* ia valoare ca și mai sus, astfel la ieșire este transmis modului *Say* următorul string “Today is d of m, y” (d- day, m-month, y-year).

CONCLUZII

Obiectivul principal al acestei teze a fost designul și implementarea unui Sistem Complet de Automatizare, exemplificarea practică a acestuia făcându-se prin intermediul unui sistem de recunoaștere automată a vorbirii și a robotului NAO. În esență, robotul trebuie să fie capabil să răspundă unor comenzi predefinite în limba română prin executarea lor. Pentru atingerea acestui scop s-au avut în vedere câteva ținte principale: înțelegerea principiilor conceptuale ale sistemului de recunoaștere automată a vorbirii implementat de echipa laboratorului SpeeD, a protocolului impus de aplicația de RAV între un client și serverul pe care aceasta rulează și determinarea soluției optime de implementare și programare a comportamentului robotului NAO. Schema bloc funcțională a Sistemului Complet de Automatizare concepută de autor este cea ilustrată în figura 0.1 din capitolul de Introducere.

Cel de-al doilea obiectiv al acestei lucrări a constat în realizarea unei arhitecturi de comunicație optime între robot și server astfel încât legătura între cele două entități să fie constrânsă doar de restricțiile impuse de protocolul de comunicație utilizat de aplicația de transcriere audio-text. Structura acestei soluții a fost prezentată în figura 3.2.1 (Capitolul 3), unde se poate observa că legătura între NAO și server se face prin Internet, fiind o legătură de tipul punct-la-punct. Pentru implementarea acestei variante autorul a efectuat un număr de setări particulare ale robotului.

Această teză scrisă prezintă etapele parcurse de autor în vederea atingerii obiectivului principal. Capitolul 1 prezintă o scurtă descriere a caracteristicilor importante ale robotului NAO atât din punct de vedere hardware, cât și software. De asemenea, au fost subliniate și relațiile între microcontrolerele principale și dispozitivele comandate de acestea, alături de particularitățile software ale robotului și limbajele de programare suportate. Autorul acestei lucrări a trebuit să determine principiul de funcționare a robotului NAO astfel încât să poată fi dezvoltat un comportament adecvat.

Contribuția personală a autorului este evidențiată în capitolele 3 și 4. Capitolul 3 se axează pe utilizarea aplicației de transcriere audio-text. Această aplicație este implementată folosind modelul client - server, ceea ce implică un protocol de comunicație bine stabilit. Autorul a trebuit, la acest nivel, să utilizeze aplicația în vederea extragerii setului de reguli impus, pentru a le putea implementa mai departe pe robot. Prin efectuarea unor pași specifici, autorul l-a făcut pe robotul NAO client al acestei aplicații.

Capitolul 4 este dedicat comportamentului robotului, mai exact realizării software a codurilor ce determină funcționarea efectivă a lui NAO. De asemenea, autorul a stabilit o listă de nouă comenzi simple, unele din ele implică mișcări ale robotului, altele apeluri ale unor funcții de bază ale sistemului de operare. Această listă a fost creată astfel încât să includă cuvinte cât mai diferite pentru o rată de transcriere oferită de sistemul RAV cât mai bună. Ca variantă de verificare, autorul a implementat o metodă prin care robotul traduce comanda din limba română atunci când aceasta este transcrisă cu succes și o alta prin care NAO pronunță că s-a produs o eroare, în cazul unei transcrieri greșite. Prin această manieră autorul are un feedback asupra acțiunilor petrecute în sistem. Implementările modulelor, ce constituie comportamentul robotului ca un tot unitar, sunt descrise în detaliu pe parcursul acestui capitol.

Toate etapele parcurse și efectuate de autor pentru realizarea practică au avut ca obiectiv crearea Sistemului Complet de Automatizare. Acesta poate fi privit ca un sistem complet deoarece înglobează în componența sa entități complexe ale altor sisteme și se comportă ca unul de sine stătător. Automatizarea acestuia este susținută de faptul că autorul a implementat comportamentul descris astfel încât, la pornirea hardware a robotului (prin apăsarea butonului de pe piept), acesta să ruleze implicit pe sistemul de operare al lui NAO.

Rezumatul contribuțiilor personale ale autorului poate fi următorul:

- Conceperea structurii Sistemului Complet de Automatizare;
- Stabilirea și implementarea arhitecturii optime de comunicație robot – server, conform protocolului folosit de aplicația de recunoaștere automată a vorbirii;
- Elaborarea listei predefinite de comenzi, utilizată pentru exemplificarea practică;
- Prelucrarea răspunsului primit de la aplicația de recunoaștere automată a vorbirii;
- Stabilirea comportamentului robotului conform comenzilor recepționate – dezvoltarea programului software necesar controlului.

Soluția sistemului propus de autor este una realizabilă practic, după cum se poate dovedi în demonstrație, însă a fost creată ca o dovadă a unui concept mult mai amplu. Cu siguranță, pot fi aduse o multitudine de îmbunătățiri. Domeniul recunoașterii automate a vorbirii este unul în plină ascensiune, mai ales cel pentru limba română. De aceea, sistemul descris în această teză poate veni în sprijinul aplicațiilor bazate pe RAV. De asemenea, acesta poate face parte dintr-un concept mult mai elaborat cum ar fi acela de "casă inteligentă". Pentru a se ajunge la asemenea performanțe, sistemului complet de automatizare realizat cu robotul NAO poate avea următoarele optimizări ce vor face obiectul altor proiecte viitoare:

- Dezvoltarea unui comportament mai complex, astfel încât robotul NAO să fie în continuă ascultare, iar la sesizarea unei comenzi vocale specifice să pornească înregistrarea, implicit comunicația cu serverul pe care rulează aplicația RAV;
- Implementarea unui sistem de recunoaștere automată a vorbirii în limba română direct pe robotul NAO, fapt ce ar duce la complexitatea comportamentului dar la facilitarea arhitecturii de comunicație existente, eliminând astfel modelul de client-server;
- Sintetizarea de voce pentru limba română astfel încât robotul NAO să fie capabil să pronunțe corect cuvintele.

BIBLIOGRAFIE

[1] https://www.aldebaran.com/en/humanoid-robot/nao-robot (accesat la 05. 06. 2015)

[2] http://en.wikipedia.org/wiki/Humanoid_robot (accesat la 06. 06. 2015)

[3] Pfiefer, R., Scheier, C., "Understanding Intelligence", Bradford Books, 2011 , https://books.google.de/books?id=iIv6-rxTCkoC&redir_esc=y (accesat la 06. 06. 2015)

[4] http://www.intel.com/content/www/us/en/processors/atom/atom-z540-z530-z520-z510-z500-45-nm-technology-datasheet.html (accesat la 07. 06. 2015)

[5] http://ark.intel.com/products/35463/Intel-Atom-Processor-Z530-512K-Cache-1_60-GHz-533-MHz-FSB (accesat la 07. 06. 2015)

[6] http://doc.aldebaran.com/2-1/home_nao.html (accesat la 08. 06. 2015)

[7] https://ro.wikipedia.org/wiki/Grad_de_libertate (accesat la 08. 06. 2015)

[8] http://users.utcluj.ro/~elupu/ASRSV/L1.pdf (accesat la 11. 06. 2015)

[9] Burileanu, D., Contribuţii la sinteza vorbirii din text pentru limba română. Teză doctorat, 1999

[10] Lect. Horia Cucu, Proiect de cercetare-dezvoltare în Tehnologia Vorbirii

[11] Renals S., Hain, T., "Speech Recognition," Handbook of Computational Linguistics and Natural Language Processing, 2010

[12] Cucu, H., "Towards a speaker-independent, large-vocabulary continuous speech recognition system for Romanian," Teză de doctorat, Universitatea Politehnica din Bucureşti, România, 2011, http://speed.pub.ro/academic/research-and-development-project-in-spoken-language-technology (accesat la 13. 06. 2015)

[13] https://en.wikipedia.org/wiki/Client%E2%80%93server_model (accesat la 17. 06. 2015)

[14] https://en.wikipedia.org/wiki/Server_(computing) (accesat la 17. 06. 2015)

[15] http://www.calculatoareinternet.ro/Arhitectura_client-server.html (accesat la 17. 06. 2015)

[16] http://vechi.upg-ploiesti.ro/col/ldumitrascu/html/content.jsp-id=462.htm#1 (accesat la 19. 06. 2015)

[17] http://www.cs.ubbcluj.ro/~rlupsa/edu/retele-2003/c5.html (accesat la 23. 06. 2015)

[18] https://ro.wikipedia.org/wiki/Paravan_(software) (accesat la 24. 06. 2015)

[19] http://doc.aldebaran.com/1-14/software/choregraphe/objects/script_box.html (accesat la 26. 06. 2015)

[20] http://doc.aldebaran.com/1-14/software/choregraphe/objects/box_input_output.html (accesat la 26. 06. 2015)

ANEXA 1

Achiziție

--If pressed filter

```
class MyClass(GeneratedClass):
def __init__(self):
GeneratedClass.__init__(self)

def onLoad(self):
pass

def onUnload(self):
pass

def onInput_onStart(self, p):
if(p > 0):
self.onStopped()

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

--Record Sound

--Get File Name

```
def onLoad(self):
self.framemanager = ALProxy("ALFrameManager")

def onInput_onStart(self):
path = os.path.expanduser('~') + "/liveTranscriber/data/test"
self.onStopped( path )
```

--Rec. Sound File

```
def onInput_onStart(self, prefix):
if(self.bIsRunning):
return
self.bIsRunning = True
self.filepath = prefix + ".wav"
if self.ad:
channels = [0,1,0,0] # 0->not active ; 1->active
            # | | | |
            # | | | - Back
            # | | --- Front
            # | ----- Right
            # ------- Left
# startMicrophonesRecording(filepath, encoding, sample_rate, channels)
self.ad.startMicrophonesRecording( self.filepath, "wav", 16000, channels )
self.bIsRecording = True
else:
self.logger.warning("No sound recorded")
```

Comandă și control

--Comandă

```
import os

class MyClass(GeneratedClass):
def __init__(self):
GeneratedClass.__init__(self)

def onLoad(self):
self.bIsRunning = False
self.posture = ''
self.text = ''
```

```
self.command = ''

def onUnload(self):
self.bIsRunning = False

def onInput_onStart(self):
if( self.bIsRunning ):
return
self.bIsRunning = True
f = open( '/home/nao/liveTranscriber/data/transcription.out', 'r' )
self.command = f.readline()
f.close()
self.command = "".join(ch for ch in self.command if ch.isalnum())
f = open( '/home/nao/liveTranscriber/data/test.out', 'w' )
f.write( self.command )
f.close()

if self.command == 'staijos' :
self.jos()
elif self.command == 'ridictenpicioare' :
self.sus()
elif self.command == 'mergiunmetru' :
self.mergi()
elif self.command == 'micmnadreapt' :
self.mana()
elif self.command == 'fiochiiroii' :
self.ochi()
elif self.command == 'fiochiialbatrii' :
self.ochi()
elif self.command == 'fiochiiverzi' :
self.ochi()
elif self.command == 'cteceasul' :
self.time()
elif self.command == 'cedateazi' :
self.date()
elif self.command == 'vorbetemaitare' :
self.volume()
elif self.command == 'vorbetemaincet' :
self.volume()
else :
self.error()

self.onStopped()
self.onUnload()


def jos(self):
self.text = 'Sit down!'
self.posture = 'Crouch'
self.onJos([self.text,self.posture,self.command])

def sus(self):
self.text = 'Get up!'
self.posture = 'Stand'
self.onSus([self.text,self.posture,self.command])

def mergi(self):
self.text = 'Walk!'
self.posture = 'Stand'
self.onMergi([self.text,self.posture,self.command])

def error(self):
self.text = 'Error!'
self.posture = ''
```

```
self.onError([self.text,self.posture,self.command])

def mana(self):
self.text = 'Hand!'
self.posture = 'Stand'
self.onMana([self.text,self.posture,self.command])

def ochi(self):
self.text = 'Eyes!'
self.posture = ''
self.onOchi([self.text,self.posture,self.command])

def time(self):
self.text = 'Time!'
self.posture = ''
self.onTime([self.text,self.posture,self.command])

def date(self):
self.text = 'Date!'
self.posture = ''
self.onDate([self.text,self.posture,self.command])

def volume(self):
self.text = 'Volume!'
self.posture = ''
self.onVolume([self.text,self.posture,self.command])

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

--Get Movement

```
class MyClass(GeneratedClass):
def __init__(self):
GeneratedClass.__init__(self)

def onLoad(self):
self.command = ''
self.color = ''

def onUnload(self):
#put clean-up code here
pass

def onInput_onStart(self, p):
self.command = p
if self.command == "mergiunmetru":
self.onMergi()
elif self.command == "micmnadreapt":
self.onMana()
elif self.command == "fiochiiroii":
self.color = 'red'
self.onOchi(self.color)
elif self.command == "fiochiiverzi":
self.color = 'green'
self.onOchi(self.color)
elif self.command == "fiochiialbatrii":
self.color = 'blue'
self.onOchi(self.color)
elif self.command == "cteceasul":
self.onTime()
elif self.command == "cedateazi":
self.onDate()
elif self.command == "vorbetemaincet":
self.onVolumeDown()
```

```
elif self.command == "vorbetemaitare":
self.onVolumeUp()

def onInput_onStop(self):
self.onUnload() #it is recommended to reuse the clean-up as the box is stopped
self.onStopped() #activate the output of the box
```

--Set Speaker Volume

```
class MyClass(GeneratedClass):
def __init__(self):
GeneratedClass.__init__(self, False)

def onInput_onVolumeUp(self):
try:
self.audiodevice = ALProxy("ALAudioDevice")
volume = self.audiodevice.getOutputVolume() + 10
self.audiodevice.setOutputVolume(volume)
except Exception as e:
self.logger.error(e)

self.onReady() # activate output of the box

def onInput_onVolumeDown(self):
try:
self.audiodevice = ALProxy("ALAudioDevice")
volume = self.audiodevice.getOutputVolume() - 10
self.audiodevice.setOutputVolume(volume)
except Exception as e:
self.logger.error(e)

self.onReady() # activate output of the box
```

--Get Date

```
import datetime

class MyClass(GeneratedClass):
def __init__(self):
GeneratedClass.__init__(self, False)

def onLoad(self):
self.string = ''
pass

def onUnload(self):
#~ puts code for box cleanup here
pass

def onInput_onGetTime(self):
currentTime = datetime.datetime.now()
self.string = "It's hour " + str(currentTime.hour+3) + " and " +
str(currentTime.minute) + " minutes!"
self.onTime(self.string)

def onInput_onGetDate(self):
currentTime = datetime.datetime.now()
month = {
1 : "January",
2 : "February",
3 : "March",
4 : "April",
5 : "May",
6 : "June",
7 : "July",
8 : "August",
```

```
9 : "September",
10 : "Octomber",
11 : "November",
12 : "December"
}
self.string = "Today is " + str(currentTime.day) + " of " +
month[currentTime.month] + ', ' + str(currentTime.year) + "!"
self.onTime(self.string)
```