Universitatea “Politehnica” din Bucureşti

Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei

SERVICIU WEB PENTRU RECUNOAȘTEREA AUTOMATĂ A VORBITORULUI (SPEAKER RECOGNITION WEB-SERVICE)

Proiect de diplomă

prezentat ca cerinţă parţială pentru obţinerea titlului de

Inginer în domeniul *Electronică şi Telecomunicaţii*

programul de studii de licenţă *Electronică Aplicată*

Conducător ştiinţific

As.Dr.Ing. HORIA CUCU

Absolvent

Radu-AlexandruHARAPU

2014

Anexa 1

Universitatea "Politehnica" din Bucureşti
Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei
Departamentul Electronica Aplicata si Ingineria Informatiei

Aprobat Director de Departament:
Prof. Dr. Ing .Sever Pasca

TEMA PROIECTULUI DE DIPLOMĂ
a studentului Harapu S. Radu-Alexandru, grupa 441B

1. Titlul temei: Serviciu web pentru recunoasterea automata a vorbitorului (Speaker recognition web-service)

2. Contribuţia practică, originală a studentului va consta în (*în afara* părţii de documentare):

Preluarea si prelucrarea semnalului vocal:
- Achizitionarea semnalului vocal prin intermediul unui microfon
- Extragerea parametrilor mfcc
- Adaugarea atat a semnalului vocal cat si a parametrilor la o baza de date
- Crearea unui model acustic universal si a modelelor acustice pentru fiecare vorbitor in parte

Recunoasterea sau verificarea vorbitorului
- Antrenarea sistemului cu modelele stocate
- Decodarea esantionului vocal primit si compararea cu modelele sistemului
- Incorporarea intregului proces intr-o aplicatie de tip client-server

3. Proiectul se bazează pe cunoştinţe dobîndite în principal la următoarele 3-4 discipline: Programare obiect orientata, Structuri de date si algoritmi, Arhitectura microprocesoarelor

4. Realizarea practică/ proiectul rămân în proprietatea: Laboratorul de Cercetare SpeeD, student Harapu Radu

5. Proprietatea intelectuală asupra proiectului aparţine: Laboratorul de Cercetare SpeeD, student Harapu Radu

6. Locul de desfăşurare a activităţii: UPB

7. Data eliberării temei: iunie 2013

CONDUCĂTOR LUCRARE: *Lect.Horia Cucu*

STUDENT: *Harapu Radu-Alexandru*

Harapu

Anexa 5

Declaratie de onestitate academica

Prin prezenta declar ca lucrarea cu titlul *Serviciu web pentru recunoșterea automata a vorbitorului* (Speaker recognition web-service), prezentata în cadrul Facultatii de Electronica, Telecomunicatii si Tehnologia Informatiei a Universitatii "Politehnica" din Bucuresti ca cerinta partiala pentru obtinerea titlului de *Inginer/ Master* în domeniul *Inginerie electronica si telecomunicatii*, programul de studii *Electronica aplicata* este scrisa de mine si nu a mai fost prezentata niciodata la o facultate sau institutie de învatamânt superior din tara sau strainatate.

Declar ca toate sursele utilizate, inclusiv cele de pe Internet, sunt indicate în lucrare, ca referinte bibliografice. Fragmentele de text din alte surse, reproduse exact, chiar si în traducere proprie din alta limba, sunt scrise între ghilimele si fac referinta la sursa. Reformularea în cuvinte proprii a textelor scrise de catre alti autori face referinta la sursa. Înteleg ca plagiatul constituie infractiune si se sanctioneaza conform legilor în vigoare.

Declar ca toate rezultatele simularilor, experimentelor si masuratorilor pe care le prezint ca fiind facute de mine, precum si metodele prin care au fost obtinute, sunt reale si provin din respectivele simulari, experimente si masuratori. Înteleg ca falsificarea datelor si rezultatelor constituie frauda si se sanctioneaza conform regulamentelor în vigoare.

Bucuresti, *3.07.2014*

Absolvent *Radu-Alexandru HARAPU*

Harapu

(semnatura în original)

Cuprins

Lista acronimelor

EM – estimation maximization
GMM – Gaussian mixture model
HMM – Hidden Markov Model
JDK – Java Developer Kit
JVM – Java Virtual Machine
MAP – maximum a-posteriori
UBM – Universal Background Model

Introducere

Primul meu contact cu domeniul recunoașterii vocale a fost în timpul practicii de vară de la sfarșitul anului 3. Nu știam exact ce presupune și nu eram conștient încă de cât de vast este acest domeniu și cât de util poate fi. În urma unui stadiu de practică la CASIA, am început doar să înțeleg câteva din concepetele fundamentale ale recunoașterii vocale. Mi s-a părut un domeniu deosebit de interesant și cu foarte mare aplicabilitate care implică diverse cunoștințe din arii diferite. Am decis să continui lucrul la proiectul realizat până atunci și să îmbunatățesc în continuare atât designul cât și funcționalitatea acestuia, să-l transform într-o aplicație care să fie utilă, dar și ușor de folosit. Consider că domeniul recunoașterii vocale este unul în permanentă dezvoltare și eu doresc să pot participa, în orice masură, la creșterea acestuia.

Prezenta lucrare își propune descrierea unei aplicații de tip client-server care să poată realiza automat recunoașterea unui vorbitor pe baza unor modele realizate folosind mixturi Gaussiene și reținute într-o bază de date disponibilă programului. Acesta se bazează pe utilitarul open-source oferit de LIUM, iar aplicația este realizată în totalitate în limbajul de programare Java.

Lucrarea conține noțiuni teoretice despre recunoașterea vocală și programarea obiect-orientată și descrierea etapelor realizării aplicației, cât și descrierea rezultatului final.

În primul capitol am prezentat ce presupune recunoașterea vocală, diferența dintre verificare și identificare și cum se extrag trăsăturile dintr-un semnal vocal. Am arătat cum se formează un model pentru vorbitor pe baza mixturilor Gaussiene și am exemplificat procentaje tipice obținute la experimente de recunoaștere a vorbitorului.

În al doilea capitol am prezentat principiile generele ale programării obiect-orientate și in particular detalii despre limbajul Java și funcționalitatea sa, împreună cu descrierea câtorva clase din JDK frecvent utilizate în proiectul curent.

Capitolul al treilea am vorbit despre utilitarul LIUM și despre cum l-am încorporat într-o aplicație de tip client-server. Am prezentat etapele acestui proces, cu problemele întâlnite pe parcurs și rezultatele experimentelor de recunoaștere de vorbitor realizate.

În ultimul capitol am descris aplicația. Am vorbit despre clasele principale, variabilele și metodele acestora și am descris funcționalitatea fiecărei clase create în cadrul acestui proiect.

Capitolul I

Recunoașterea vorbitorului

1.1 Prezentare generală

Vorbirea oferă mai multe niveluri de informație. La un nivel de bază, vorbirea transmite cuvintele sau mesajul vorbit, dar la un nivel secundar, vorbirea oferă detalii și despre vorbitor. Lucrarea curentă se bazeaza pe o abordare statistică de modelare a vorbitorilor prin reprezentarea caracteristicilor care stau la baza vocii unei persoane utilizând mixture Gaussiene. Folosind aceste modele, se poate recunoaște vorbitorul indiferent de mesajul transmis.

Sarcini care sunt executate ușor de către oameni, precum recunoașterea fețelor sau vocilor, sunt dificil de reprodus pe calculator. Tehnologia de recunoaștere a vorbitorului iese în evidența prin faptul că este una din aplicații în care calculatorul are rezultate mai bune că omul.

Abilitatea de a recunoaște si distinge vocile a fost atent studiată timp de peste 60 ani. Prin stabilirea unor factori care dau informații dependente de vorbitor, s-a putut îmbunatați naturalețea vocii sintetizate și s-a putut evalua nivelul de încredere al recunoașterii vocale în criminalistică. La scurt timp după dezvoltarea calculatoarelor digitale, cercetarea în domeniul recunoașterii vorbitorului s-a orientat către dezvoltarea unor tehnici obiective pentru recunoaștere automată, care a dus la descoperirea că sisteme automate simple pot avea rezultate mai bune că oamenii pe sarcini similare. [1]

În ultimele decenii s-au dezvoltat algoritmi de recunoaștere a vorbitorului din ce în ce mai sofisticați si performanța acestor algoritmi în evaluarea pe situații realiste a crescut.

Sarcina generală a recunoașterii automate a vorbitorului este departe de a fi terminată si mai sunt de rezolvat multe probleme și de trecut peste multe limitari.

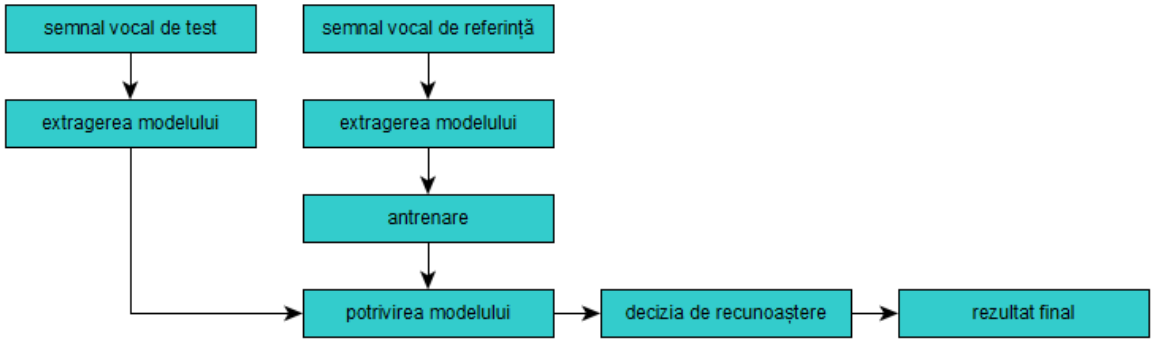

Figura 1.1 Diagrama bloc a unui sistem de recunoaștere de vorbitor

Recunoașterea vorbitorului se poate împarti în doua categorii: identificare si verificare. Pentru identificare, scopul este să se determine care voce dintr-un set cunoscut se potrivește cel mai bine vorbitorului. Pentru verificare, scopul este să se stabilească dacă vorbitorul este cine susține că este.

La identificarea vorbitorului, se presupune că vocea necunoscută aparține unui set predefinit de vorbitori cunoscuți. Pentru acest tip de problemă de clasificare (alegerea unui element dintr-un set de N) erorile sunt definite ca recunoașteri greșite, adică sistemul atribuie vocea vorbitorului unui alt vorbitor. Dificultatea indentificarii crește pe masura ce setul de vorbitori se mareste.

Aplicațiile care au doar indentificare sunt rare deoarece în situații reale este improbabil să existe doar vorbitori cunoscuți sistemului, numiți vorbitori înrolati (enrolled speakers). Totuși, o aplicație indirectă a identificarii este recunoașterea vocală adaptivă, în care vocea unui vorbitor necunoscut este cuplată cu vocea cea mai asemanatoare din setul cu care este antrenat sistemul. Alte potențiale aplicații ale identificării pot fi roboți telefonici inteligenți cu mesaj personalizat și etichetare automată a unei sedințe pentru indexare dependentă de vorbitor.

Verificarea vorbitorului presupune că sistemul decide, în funcție de un prag stabilit, dacă vocea vorbitorului corespunde cu identitatea asumată. Vorbitori cunoscuți sistemului care afirmă propria identitate se numesc pretendenți. Vorbitori cunoscuți sau necunoscuți care pretind a fi alți vorbitori se numesc impostori. Exista două tipuri de erori de verificare: acceptare falsă, în care sistemul identifică un impostor ca fiind un pretendent, si rejecție falsă, în care sistemul respinge un pretendent ca fiind un impostor.

Verificarea este de obicei la baza majorității aplicațiilor de recunoaștere a vorbitorului. Aplicațiile curente, cum ar fi tranzacțiile bancare prin telefon sau înregistrarea pe calculator (login), înlocuiesc sau adaugă la o parola memorată și verificarea vorbitorului.

Sarcinile de recunoaștere a vorbitorului pot fi împartite și în funcție de constrângerile legate de text. Într-un sistem dependent de text, vocabularul folosit se restrange la un singur cuvânt sau o frază și astfel sistemul poate profita de faptul că se spune același lucru de fiecare dată. Dar un astfel de sistem poate fi păcălit prin înregistarea frazei spuse de pretendent și redarea acesteia către sistem. Într-un sistem independent de text, vocabularul folosit la antrenare și testare nu mai este limitat. Acest tip de sistem este cel mai flexibil [2].

Între cele doua extreme ale sistemelor dependente si independete de text se află sistemele dependente de vocabular, unde ce se vorbește este restrâns la un vocabular limitat cum ar fi cifrele, folosite în aplicația curentă. Acest sistem este mai flexibil decât cel dependent de text deoarece o parolă poate fi ușor schimbata fără reantrenarea sistemului.

1.2 Trăsăturile pentru sistemele de recunoaștere a vorbitorului

Pentru a crea sisteme de recunoaștere a vorbitorului, s-a pus problema cum oamenii fac distincția între persoane doar pe baza vocii. Utilizăm mulți indicatori perceptuali, unii non-verbali, în recunoașterea vorbitorului. Acești indicatori nu sunt bine înțeleși si variaza de la unii de nivel înalt, legați de aspectele semantice sau lingvistice ale vorbirii, până la cei de nivel scăzut, care se referă la aspectele acustice ale vorbirii.

Indicatorii de nivel înalt includ cuvintele folosite, idiosincrasii în pronunțare și alte caracteristici non-acustice care pot fi atribuite unui vorbitor. Acești indicatori oferă informații despre modul de vorbire al unei persoane și se consideră că provin din experiența de viața a unei persoane, precum locul de naștere sau nivelul de educație. Indicatorii de nivel scăzut se referă la cum sună vocea unei persoane, cum ar fi dacă vorbește tare sau încet, rapid sau lent.

Deși oamenii folosesc toți acești indicatori pentru a recunoaște persoana cu care vorbesc, s-a stabilit că cel mai eficient pentru recunoașterea automată este utilizarea indicatorilor de nivel scăzut. Aceștia pot fi legați de măsurătorile acustice care sunt ușor de extras din semnalul vocal, fată de

indicatorii de nivel înalt care sunt greu de extras din semnalul vocal și pot să apară rar în vorbirea independentă de text și chiar deloc în vorbirea dependentă de text.

Pentru a găsi măsurători acustice ale semnalului vocal ce pot fi legate de atributele fiziologice ale vorbitorului, s-a utilizat modelul de bază al producerii vorbirii. Acesta spune că sunetele sunt produsul unui jet de aer care trece prin glotă, producând rezonanțe în tractul vocal și cavitățile nazale. În timpul sunetelor sonore, cum ar fi vocalele, glota se deschide și se închide ritmic pentru a produce o excitare a tractului vocal. În timpul sunetelor nesonore, cu ar fi fricativele, glota rămâne parțial deschisă și crează o excitare prin turbulența aerului. Pentru a produce diferite sunete, tractul vocal se mută în diverse configurații care își schimbă structura de rezonanță. Sunetele nazale sunt produse prin trimiterea excitației glotei prin cavitățile nazale.

Din acest model putem vedea că glota si tractul vocal oferă majoritatea caracteristicilor dependente de vorbitor din semnalul vocal. Periodicitatea semnalului vocal ne dă informații despre glotă. Componentele majore ale spectrului vocal conțin informații despre tractul vocal și cavitatea nazală. Dintre acestea, componentele de frecvența s-au dovedit a fi cele mai eficiente pentru recunoșterea automată a vorbitorului. Deși periodicitatea oferă și ea informații legate de vorbitor și poate fi folosită în anumite aplicații controlate atent, poate fi dificil de extras corect, mai ales în medii cu zgomot, și este afectată de factori nefiziologici, de exemplu starea emoțională a vorbitorului. [3]

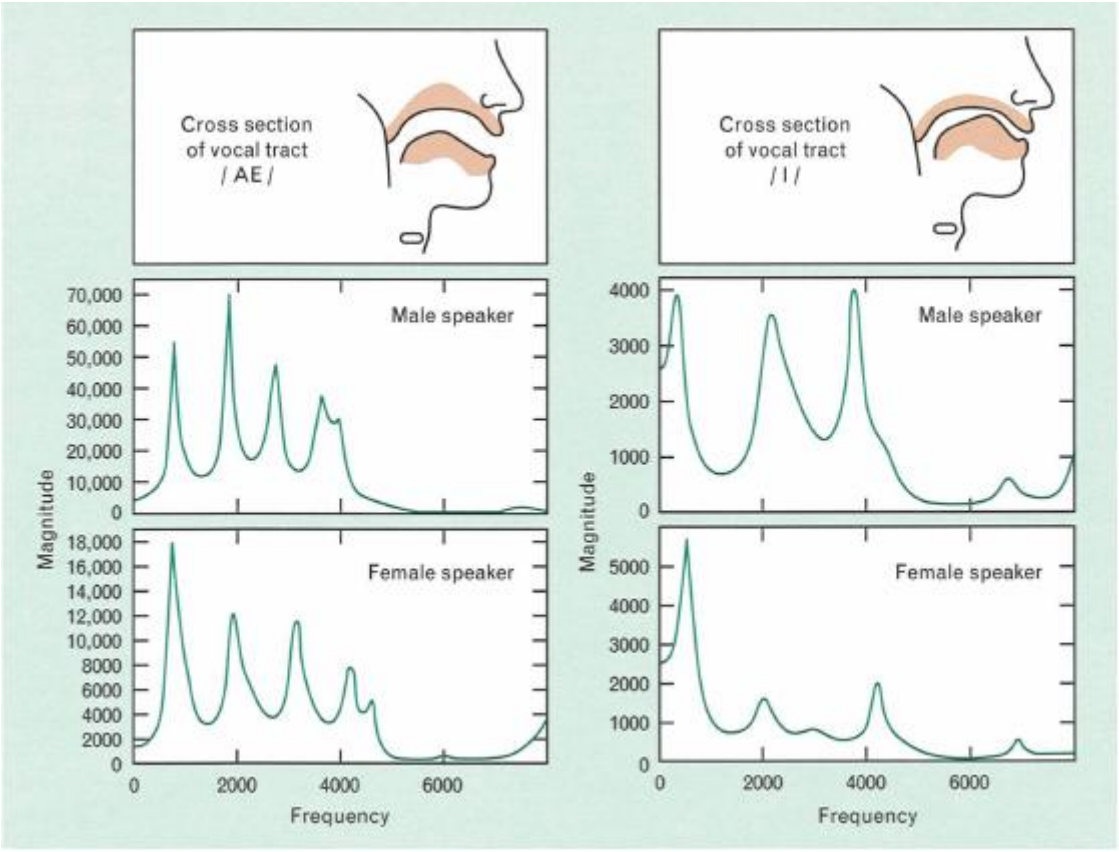

Figura 1.2 Configurații ale tractului vocal si spectrele corespunzătoare pentru vorbitori de sex masculin si feminin

În figura se pot observa exemple ale configurațiilor tractului vocal și cum acestea produc spectre diferite pentru două vocale diferite. Partea de sus este o secțiune transversală a tractului vocal, iar partea de jos este o reprezentare grafică a spectrului de frecvența. Vârfurile din spectru

sunt rezonanțe produse de acea configurație particulară și sunt cunoscute că formanți. Pentru fiecare configurație, avem doi vorbitori, unul de sex masculin și unul de sex feminin.

Pentru același sunet, formanții au aproximativ aceeași poziție pe grafic indiferent de vorbitor. Totuși, compârand graficele, se observă diferențe de frecvență si intensitate rezultate din structurile tractului vocal. Majoritatea sistemelor de recunoaștere automată a vorbitorului se bazează pe aceste diferențe pentru a distinge între vorbitori. [4]

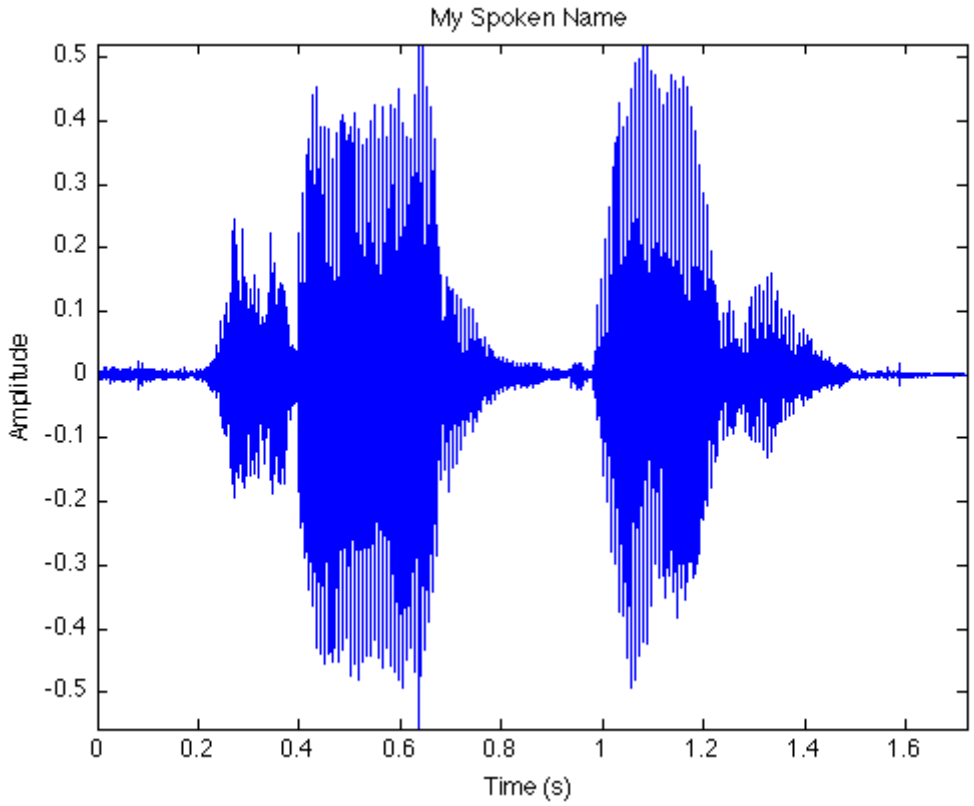

Figura 1.3 Forma de undă a unui semnal vocal

Vorbirea normală nu este doar o concatenare a sunetelor, ci o îmbinare a lor, adesea fără limite exacte între schimbări. Figura de mai sus este un exemplu de semnal vocal eșantionat digital si reprezintă forma de undă a unei propoziții spuse în vorbire continuă.

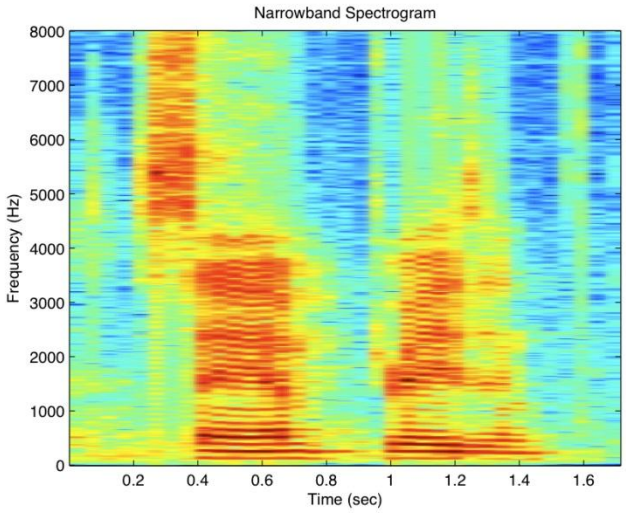

Figura 1.4 Spectrograma unui semnal vocal

În figura 3 este o spectrogramă, care este reprezentată în trei dimensiuni, pe axa x fiind timpul, pe axa y frecvența si regiunile mai întunecate fiind zone cu energie spectrală mai mare. Spectrograma ilustrează natura dinamică a formanților.

Pentru a obține măsuratori stabile ale spectrului din vorbire continuă, se executa analiza spectrală în timp scurt, care implică mai mulți pași. Întâi vorbirea este segmentată în ferestre de 20-40ms, standardul fiind de 25ms. Diferența dintre ferestre este de obicei 10ms, ceea ce permite o suprapunere parțiala.

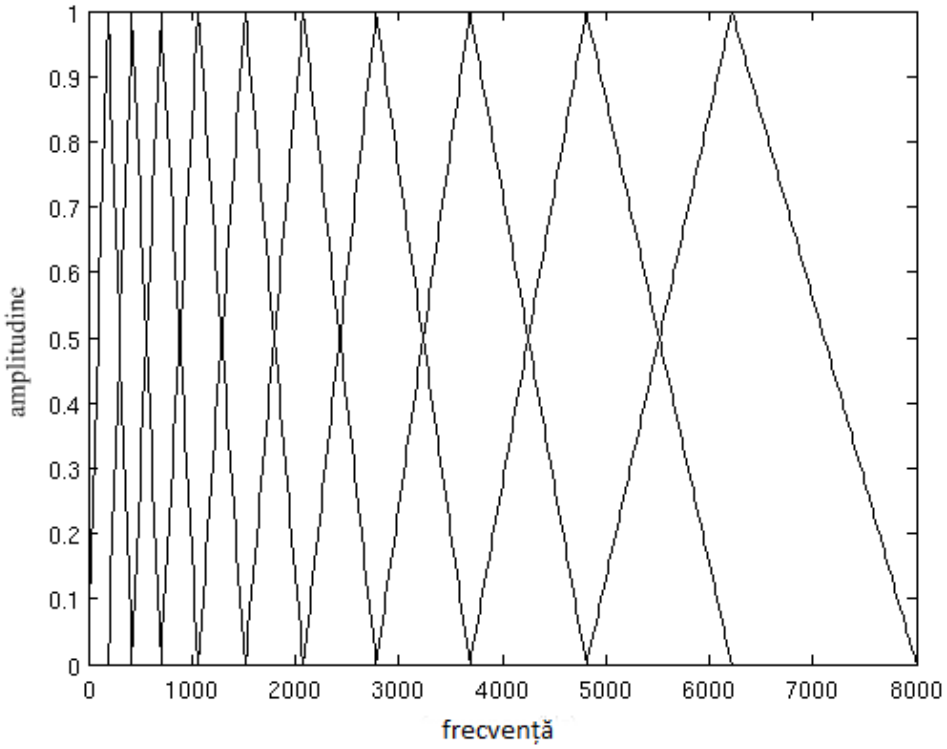

Figura 1.5 Banc de filtre Mel

În continuare, urmatorii pași sunt aplicați fiecărei ferestre și se extrag trăsăturile spectrale, câte un set de 12 coeficienți pentru fiecare fereastră

Se aplică transformata Fourier rapidă și obținem estimatul bazat pe periodogramă al puterii spectrale. Din aceasta luăm valoarea absolută a transformatei Fourier complexe si ridicăm rezultatul la patrat. În general se face o transformată Fourier în 512 puncte si se păstrează doar primii 257 coeficienți.

Apoi se crează bancul de filtre Mel. Acesta e un set de 20 până la 40 de filtre triunghiulare, standardul fiind 26, și se aplică asupra estimatului bazat pe periodogramă al puterii spectrale de la pasul anterior. Pentru a calcula energiile bancului de filtre înmulțim fiecare banc de filtre cu puterea spectrală și apoi se adună coeficienții. Odată ce ce s-a facut această operație obținem 26 de numere care ne spun câtă energie se găsea în fiecare banc de filtre. Acestora li se aplică funcția logaritm și rezultatului i se aplică mai departe transformata cosinus discretă (Discrete Cosine Transform – DCT) din care rezultă cele 12 trăsături pentru fiecare fereastră numite coeficienți mel-cepstrali (Mel Frequency Cepstral Coefficients – MFCC) [5].

Se mai pot utiliza si coeficienții delta și delta-delta, numiți si coeficienți diferențial și de accelerație. Vectorul de trăsături MFCC descrie doar anvelopa putereii spectrale a unei singure ferestre. Prin calcularea traiectoriilor și adăugarea lor la vectorul original obținem informații despre cum se schimbă coeficienții MFCC în timp si duce la performanțe mai bune în recunoașterea automată a vorbitorului.

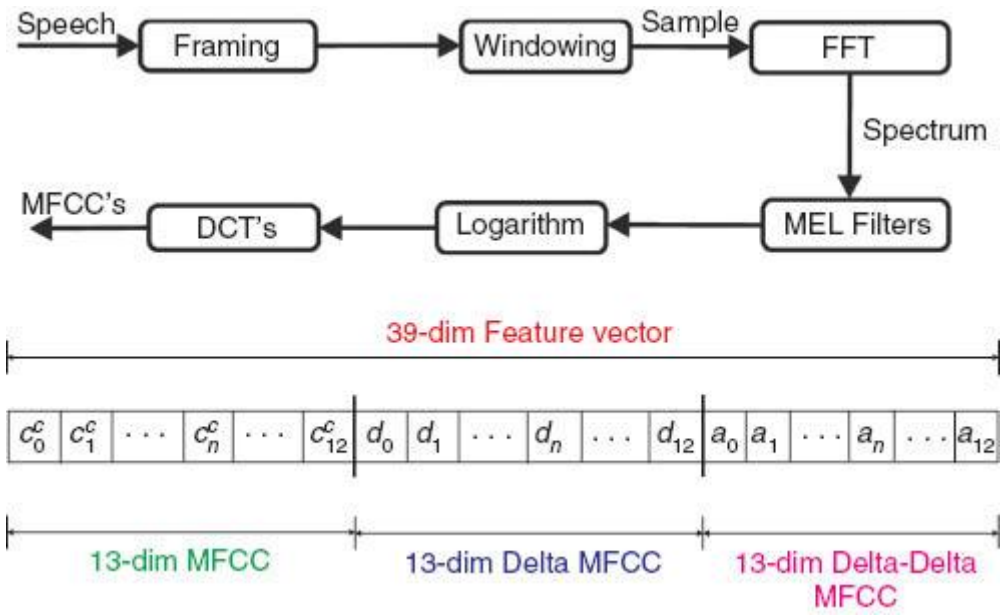

Figura 1.6 Procesul de extragere a vectorilor de trăsături.

Vectorii de trăsături care rezultă în urma acestui proces este foarte adesea punctul de plecare pentru orice aplicație legată de semnalul vocal, incluzând recunoașterea vocală si identificarea limbii. Din păcate, această reprezentare nu este întotdeauna cea mai eficientă pentru recunoașterea vorbitorului. O mare parte a secvenței spectrale reprezintă conținutul lingvistic care include multe redundanțe și de care în mare parte nu avem nevoie pentru recunoașterea vorbitorului.

1.3 Modelul statistic al vorbitorului

Sarcini specifice de recunoaștere a vorbitorului sunt îndeplinite prin utilizarea unor modele care extrag si reprezintă informația dorită din secvența spectrală. Din moment ce informația dependentă de vorbitor principală oferită de spectru este despre forma tractului vocal, dorim să

folosim un model care să indice formele tractului vocal caracteristice vocii unei persoane, așa cum se manisfestă ele în trăsăturile spectrale. Datorită succesului recunoșterii statistice a modelelor pentru o gamă largă de aplicații legate de vorbire, s-a adaptat formularea statistică pentru generarea unui astfel de model al vorbitorului.

Pentru modelul statistic, se consideră vorbitorul că fiind o sursă aleatoare ce produce vectori de trăsături. În interiorul sursei se găsesc stări ascunse ce corespund configurațiilor caracteristice ale tractului vocal. Când o sursă aleatore este într-o stare particulară, produce vectori de trăsături corespunzatori acelei configurații particulare a tractului vocal. Aceste stări se numesc ascunse doarece putem să observăm doar vectorii care rezultă, nu și stările care stau la baza lor.

Fiindcă semnalul vocal nu este un semnal determinist și spectrul produs de o anumită formă a tractului vocal poate varia mult datorită efectelor de coarticulare, fiecare stare generează vectori de trăsături corespunzătoare unei funcții de densitate de probabilitate Gaussiene multidimensională, cu o medie si covarianța dependente de stare.

Pe lângă faptul că vectorul de trăsături este produs de o sursă aleatore dependentă de stare, procesul care indică în ce stare se afla modelul vorbitorului la orice moment de timp este și el un proces aleator. Funcția de densitate de probabilitate discretă asociată celor M stări care descrie probabilitatea de a se afla în orice stare este $\{p_1, p_2, ..., p_M\}$, unde

$$\sum_{i=1}^{M} pi = 1$$

și funcția de densitate de probabilitate discretă care descrie probabilitatea că o tranziție dintr-o stare în alta să aibă loc este:

$$a_{ij} = Pr(i \rightarrow j), \text{ pentru } i,j = 1,2,...M$$

Definiția de mai sus a modelului statistic este cunoscut sub denumirea de model Markov ascuns ergodic (Hidden Markov Model – HMM).

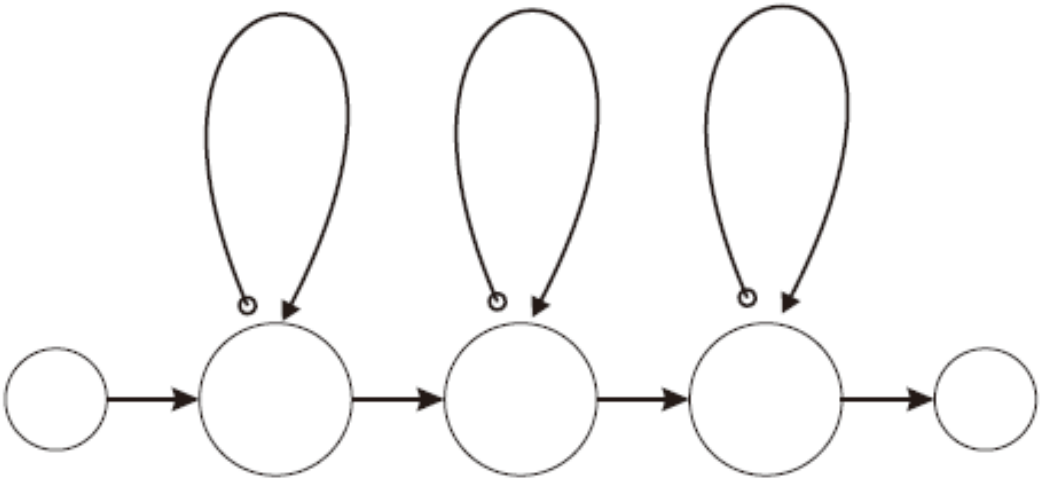

Figura 1.7 Exemplu de topologie a unui model Markov ascuns

HMM-urile au o fundație teoretică bogată și au fost aplicate extensiv într-o varietate de sarcini de identificare statistică a modelelor, în domeniul recunoașterii vocale și în altele. Motivul principal pentru utilizarea HMM-urilor în recunoașterea vocală este că oferă un model structurat,

flexibil și maleabil din punct de vedere computațional care să descrie un proces statistic complex. [6]

1.4 Modelul de vorbitor bazat pe mixturi Gaussiene

Din definiția de mai sus a modelului statistic al vorbitorului, se poate arăta că funcția de densitate de probabilitate a trăsăturilor generate de un model statistic al vorbitorului este o mixtură Gaussiană (Gaussian Mixture Model – GMM).

Funcția de densitate de probabilitate a unei mixturi Gaussiene este

$$p(x|\lambda) = \sum_{i=0}^{M} pi\ bi(x)$$

unde

$\lambda = (p_i, \mu_i, \sum_i)$, pentru i = 1,...M

μ_i = vectorul de medie

$\sum_i$ = matricea de covarianță

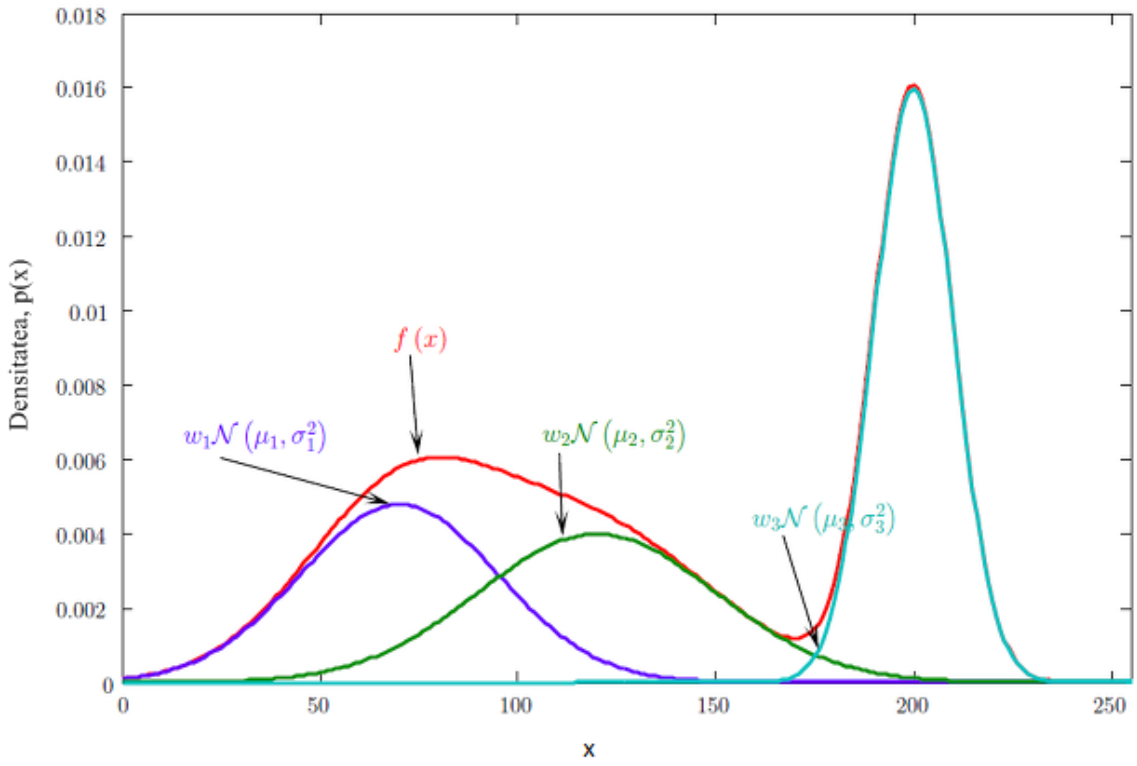

Figura 1.8 Exemplu de mixtură Gaussiana

Și reprezintă parametrii modelului de vorbitor. Astfel probabilitatea că un vector de trăsături X_t provenit de la un model de vorbitor cu parametrul λ este suma probabilităților că X_t a fost generat din fiecare stare ascunsă, ponderată cu probabilitatea de a fi în fiecare stare. Cu această sumă putem obține o valoare cantitativă, sau scor, pentru posibilitatea că un vector necunoscut de trăsături să fi fost generat de un anume model de vorbitor bazat pe GMM.

Deși la prima vedere mixturile Gaussiene par săaibă o complexitate ridicată, estimate ale parametrilor modelului sunt obținute nesupervizat prin utilizarea algoritmului expectation-maximization – EM. Algoritmul EM preia vectori de trăsăsturi obținuți din semnalul vocal de antrenare al unui vorbitor și prelucrează iterativ estimatele parametrilor modelului pentru a maximiza probabilitatea că modelul să corespundă distribuției datelor de antrenare. Această antrenare nu are nevoie de informații suplimentare, cum ar fi transcrierea a ce s-a vorbit, și paramterii converg către o soluție finală în doar câteva iterații. [7]

1.5 Aplicarea modelului

Folosind GMM-ul că reprezentare de bază a unui vorbitor, putem să aplicăm acest model sarcinilor specifice de identificare si verificare. Sistemul de identificare este unul simplu, de clasificare în funcție de similaritatea maximă. Pentru un grup de referință conținând S vorbitori $\{\lambda_1, \lambda_2, \ldots \lambda_S\}$, obiectivul este să se gasească identitatea vorbitorului al cărui model are probabilitatea maximă pentru secvența de vectori de trăsături $X = \{x_1, x_2, \ldots x_T\}$.

Deși decizia de verificare este una binară, este mai dificil de facut decât identificarea deoarece variantele sunt mai slab definite. Sistemul trebuie să decidă dacă vocea aparține vorbitorului pretendent, având un model bine definit, sau altui vorbitor, având un model prost definit.

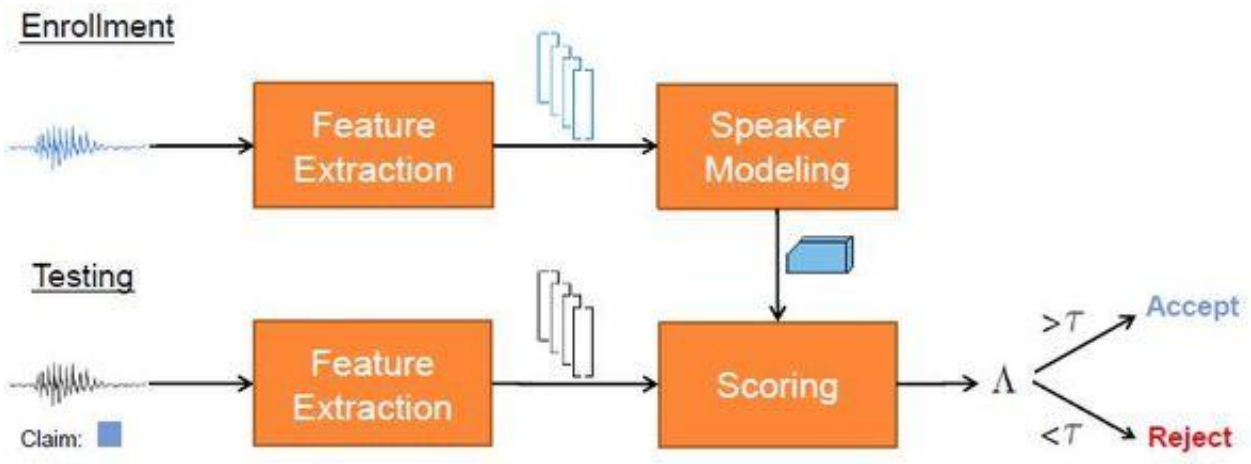

Figura 1.9 Model general al verificării vorbitorului

Într-un scenariu de testare, pentru un enunț dat și o identitate revendicată, decizia devine H_0, daca X aparține vorbitorului declarat sau H_1, daca X nu aparține.

Un model al univerului de vorbitori nerevendicați trebuie să fie folosit pentru a executa testul de maximă probabilitate care decide între H_0 si H_1. Abordarea generală în cazul de verificare a vorbitorului este să se afle raportul de probabilitate pentru enunțul dat la intrarea sistemului care să determine dacă vorbitorul este acceptat sau respins. Pentru un enunț X, o identitate revendicată cu modelul corespondent λ_C si modelul de vorbitori posibili nerevendicați $\lambda_{C'}$, raportul de probabilitate este

$$\frac{\Pr(X \text{ apartine vorbitorului revendicat})}{\Pr(X \text{ nu apartine vorbitorului revendicat})} = \frac{\Pr(\lambda C|X)}{\Pr(\lambda C'|X)}$$

Dacă aplicăm criteriul lui Bayes și eliminăm posibilitățile anterioare pentru vorbitor pretendent și impostor, raportul de probabilitate în domeniul logaritmic devine

$$\Lambda(X) = \log p(X|\lambda_C) - \log p(X|\lambda_{C'})$$

Termenul $p(X|\lambda_C)$ este probabilitatea că enunțul să aparțină vorbitorului pretendent și $p(X|\lambda_{C'})$ este probabilitatea că enunțul să nu aparțină vorbitorului pretendent. Raportul de probabilitate este comporat cu un prag θ, vorbitorul fiind acceptat daca $\Lambda(X) \geq \theta$ și respind daca $\Lambda(X) \leq \theta$. Raportul de probabilitate măsoară cu cât este mai bun modelul pretendentului față de un model al unui vorbitor nepretendent. Pragul de decizie este setat astfel încât să se ajungă la un compromis între a respinge enunțurile unui vorbitor adevărat (erori de falsă rejecție) și acceptarea vorbitorilor falși (erori de falsă acceptare). Într-o aplicație din viața reală, spre exemplu tranzacții bancare prin telefon, compromisul se face între satisfacția clienților și securitate.

Termenii raportului de probabilitate sunt calculați cu formula

$$\log p(X|\lambda_C) = \frac{1}{T}\sum_{t=1}^{T} \log p(X|\lambda c) \quad (1)$$

Factorul 1/T este utilizat pentru normalizarea probabilității pe durata enunțului.

Probabilitatea că enunțul să nu fi provenit de la vorbitorul pretendent este formată prin utilizarea unei colecții de vorbitori de fundal (background speakers). Cu un set de B modele de vorbitori de fundal $\{\lambda_1, \lambda_2, \ldots \lambda_B\}$, probabilitatea se calculează cu formula

$$\log p(X|\lambda_C) = \log\{ \frac{1}{B}\sum_{b=1}^{B} \log p(X|\lambda b) \}$$

unde $p(X|\lambda b)$ se calculează că în formula(1). $p(X|\lambda_C)$ este densitatea de probabilitate comună că un enunț să provină de la unul din vorbitorii de fundal, dacă presupunem vorbitori cu probabilitați egale.

1.6 Modelul universal de fundal (UBM)

La verificarea vorbitorului identitatea asumată este testată prin compararea scorului obținut cu modelul vorbitorului asumat și al scorului obținut cu modelul impostorilor. Modelul impostorilor este un GMM care modelează toți vorbitorii fără vorbitorul pretendent și este cunoscut sub numele de UBM (Universal Background Model). În sensul cel mai strict, pentru verificare GMM-ul impostorilor ar trebui să fie antrenat prin utilizarea datelor tuturor vorbitorilor, cu excepția vorbitorului care este verificat. În practică, UBM-ul este creat folosind datele de la toți vorbitorii, inclusiv cel verificat, presupunând că datele sunt egal distribuite pe toate subgrupurile (de exemplu, barbați și femei). Acest lucru are avantajul că UBM-ul poate fi folosit pentru verificarea oricărui vorbitor. Deoarece cantitatea de date de antrenare este foarte mare pentru UBM, nu este ieșit din comun să rezulte un număr mare de mixturi, în jurul valorii de 256 sau mai mare. Conceptual, UBM-ul reprezintă distribuția independentă de vorbitor a trăsăturilor întregii colecții de date.

Deși modelul universal poate fi folosit în identificarea cu set deschis pentru detecția vorbitorilor necunoscuți, în identificarea cu set închis nu este nevoie de un UBM deoarece GMM-urile

individuale sunt suficiente pentru procesul de identificare. Totuși, cum UBM-ul este mai bine antrenat decât orice din modelel individuale și este mai exact în modelarea întregului spațiu al trăsăturilor nu are problema insuficienței datelor sau datelor necunoscute. Modele statistice ca GMM-urile pot fi nu doar estimate direct folosind o metodă ca algoritmul EM, dar cu o cantitate mică de date paramterii pot fi adaptați noilor date folosing o metodă ca MAP (maximum a-posteriori). Astfel, o alternativă la crearea modelelor individuale este antrenarea UBM și formarea modelelor individuale prin adaptare.

Ecuația generală a algoritmului MAP este:

$$\hat{\vec{\mu}}_m = \frac{N_m}{N_m + \tau}\bar{\vec{\mu}}_m + \frac{\tau}{N_m + \tau}\vec{\mu}_m$$

unde $\hat{\vec{\mu}}_m$ este media adaptată pentru mixtura m, τ este un parametru de ponderare a cunoștințelor a priori, N_m este probabilitatea de ocupare a datelor de adaptare, $\vec{\mu}_m$ este media pentru UBM și $\bar{\vec{\mu}}_m$ este media adaptării observate.

1.7 Vorbitorii de fundal

Vorbitorii de fundal au fost folosiți cu succes în mai multe sisteme de verificare de vorbitor diferite pentru a rezulta diverse rapoarte de probabilitate. Normalizarea probabilității, dată de vorbitorii de fundal, este importantă pentru procesul de verificare deoarece ajută la minimizarea variațiilor date de alte elemente decât vorbitori la scorurile pe enunțurile de test, permițând să se aleagă un prag stabil. Scorul de probabilitate absolută al unui enunț de la un vorbitor este influențat de mulți factori care depind de enunț, incluzând conținutul lingvistic și calitatea vorbirii. Acești factori ridică dificultatea în a stabili un prag de decizie pentru valorile de probabilitate absolută care sa fie folosit în diverse situații de verificare. Identificarea nu are nevoie de normalizare deoarece deciziile sunt luate folosind scorul de probabilitate de la un singur enunț și nu necesită comparații între enunțuri diferite.

Diferența fundamentală între identificare și verificare este numărul de posibilități de decizie. La identificare, acest număr este egal cu dimensiunea populației. Ca urmare, performanțele sistemului scad pe măsură ce dimensiunea populației crește. De asemenea, identificarea are două variante, cu set deschid sau închis. Identificarea cu set deschis presupune că poate să nu existe un model pentru vorbitorul care rostește enunțul, caz în care trebuie să avem varianta de decizie "vorbitorul nu aparține setului de antrenare". Verificarea poate fi interpretată ca un caz special de identificare cu set deschis în care dimensiunea populației cunoscute este unu.

1.8 Alegerea vorbitorilor de fundal

În alegerea vorbitorilor de fundal, apar două probleme importante: alegerea vorbitorilor și numărul lor. În mod intuitiv, vorbitorii de fundal ar trebui să fie aleși să reprezinte populația impostorilor, care este în general specifică aplicației. În unele scenarii, putem presupune că impostorii o să încerce să se autentificie folosind indentitatea unui vorbitor care sună asemănător sau sunt măcar de același sex (impostori dedicați). Într-o aplicație bazată pe telefon accesibilă de un grup divers de impostori, aceștia pot să sune foarte diferit de persoanele ale căror identități își asumă (de exemplu, un vorbitor de sex masculin pretinde a fi un vorbitor de sex feminin).

Sistemele precedente s-au bazat pe alegerea vorbitorilor de fundal ale căror modele se apropie cel mai mult sau sunt cele mai competitive cu fiecare utilizator înrolat.Această alegerea este potrivită pentru cazul cu impostori dedicați, dar lasă sistemul vulnerabil la impostori cu voci diferite de ale vorbitorului a cărui identitate și-o asumă.

Deși putem folosi metode de respingere a vocilor care sunt foarte diferite pe baza stabilirii unui prag pentru raportul de probabilitate, în cazul acesta s-a mers pe varianta alegerii corespunzătoare a vorbitorilor de fundal.

Ideal, numărul de vorbitori de fundal trebuie să fie cât mai mare pentru a modela cât mai bine polulația impostorilor, dar din motive practice de timp și spațiu trebuie să se aleagă un set mic. Această dimensiune limitată este dată de puterea computațională, și ea limitată. [8]

1.9 Experimente de identificare

Vom lua în considerare experimentele făcute de laboratorul Lincoln din cadrul MIT, efectuate pe mai multe baze de date consacrate, despre care se găsesc mai multe date în Anexa 1:

- TIMIT – un corpus de vorbire tradus fonetic și lexical conținând 630 vorbitori de engleă americană, fiecare rostind câte 10 enunțuri bogate fonetic și înregistrate la microfon.
- NTIMIT – același corpus ca la TIMIT, doar că transmis prin intermediul unei rețele de comunicații și redigitizate
- Switchboard – o colecție de fișiere audio în care 113 subiecți din categorii diverse și in medii diverse participă la conversații cu durate cuprinse între cinci si șase minute utilizând telefoane GSM.

Scopul experimentelor a fost să examineze performanțele unui sistem de identificare în funcție de dimensiunea populației, atât pentru vorbirea de bandă largă, curată cât și pentru vorbirea prin intermediul telefonului. Performanțele TIMIT oferă indicații despre cât de aglomerat este spațiul vorbitorilor în condiții aproape idealea. Rezultatele de la NTIMIT indică pierderile de performanță de la folosirea vorbirii prin telefon, plină de zgomot. Rezultatele de la baza Switchboard, mult mai apropiată de realitate, oferă o măsură mai bună pentru performanțele așteptate de la utilizarea aplicațiilor de recunoaștere vocală cu convorbiri telefonice.

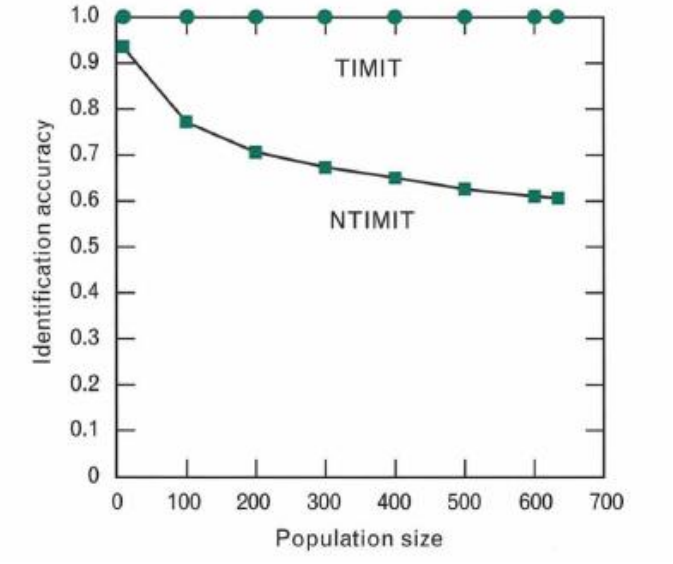

Figura 1.10 Interpretarea grafică a rezultatelor testelor cu bazele de date TIMIT și NTIMIT

Pentru experimentele pe TIMIT și NTIMIT, s-au folosit toți cei 630 vorbitori. Modelele acestora au fost realizate din GMM-uri cu 32 componente și 8 enunțuri cu o durată totală de aproximativ 24 secunde. Cele două enunțuri rămase, cu o durată aproximativa de 3 secunde fiecare au fost folosite individual ca și test.

Acuratețea identificării pentru o populație a fost calculată prin executarea repetată a unor experimente de identificare a vorbitorului pe un set de cincizeci de vorbitori aleși aleator din setul de 630 disponibili și făcând media rezultatelor. Această procedură a ajutat la a obține un scor mediu pentru o anumită compoziție a populației. Au fost folosite populații de dimensiuni 10, 100, 200, 300, 400, 500, 600 și 630.

După cum se poate observa în figura 1.9, pentru baza TIMIT, creșterea populației afectează insesizabil performanța sistemului. Acest rezultat ne spune că factorul care limitează indentificarea vorbitorului nu este aglomerarea spațiului vorbitorilor. Totuși, la NTIMIT, cu degradările liniilor telefonice apare o scădere constantă a acurateței pe măsură ce populația crește. Cea mai mare scădere a performanței are loc la trecerea de la 10 la 100 de vorbitori. După peste 200 vorbitori scăderea este aproape liniară. La utilizarea întregii populații de 630, diferența este de 39% între performanțele TIMIT și NTIMIT.

Pentru populația completă de 630, la baza TIMIT nu s-au înregistrat erori de confuzie între sexe, iar pentru NTIMIT s-au înregistrat patru astfel de erori. Acuratețea pe toată baza TIMIT a fost de 99,8% la bărbați și 99,0% la femei, iar pentru NTIMIT avem 62.5% și respectiv 56,5%.

Principalii factori care determină diferența dintre rezultatele celor două baze de date par să fie zgomotul și limitarea în bandă. TIMIT are un raport semnal-zgomot mediu de 53 dB, iar la NTIMIT este de 36 dB.

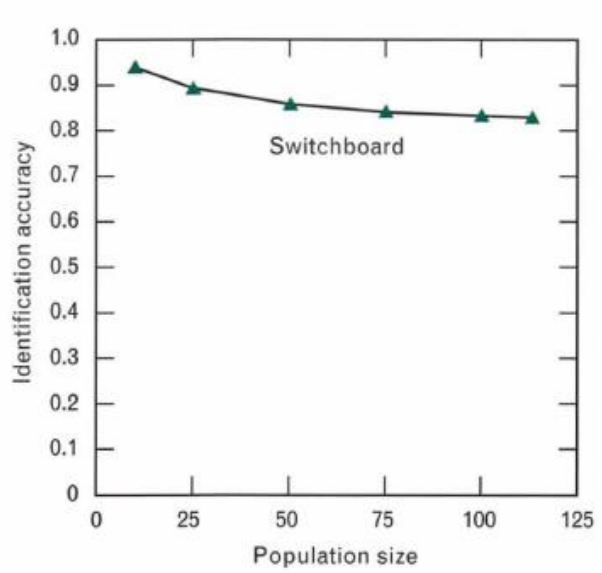

Figura 1.11 Interpretarea grafică a rezultatelor testelor cu baza de date Switchboard

Pentru Switchboard, 113 vorbitori au fost folosiți și modelați cu mixturi Gaussiene cu 64 componente. Testarea s-a făcut pe 472 de enunțuri de un minut fiecare. Avem între 2 și 12 enunțuri de test pentru fiecare vorbitor, cu o medie de 4 per vorbitor. Acuratețea identificării a fost calculată ca ai sus, doar că s-au folosit 100 seturi pentru populații de 10, 25, 50, 75, 100 și 113.

Deșinu sunt direct comparabile, rezultatele pentru Switchboard au aceeași tendință de descreștere ca la NTIMIT, dar cu o pantă nu atât de abruptă. Datorită faptului că avem mai multe

date pentru antrenare și testare și raportul semnal-zgomot este mai bun (40 dB sau mai mult), rezultatele pentru Switchboard sunt bune ca cele pentru NTIMIT. Pentru populația de 113 vorbitori, acuratețea medie a fost de 82,8%. Există două erori de identificare a unui vorbitor ca fiind de sex opus și precizia medie pentru bărbați a fost de 81,9%, respectiv 84,3% pentru femei.

1.10 Experimente de verificare

Pentru verificare, s-au folosit tot bazele de date TIMIT, NTIMIT și Switchboard la care se adaugă YOHO, o bază de date cu 138 vorbitori și enunțuri compuse din numere, 3 secvențe de numere de 2 cifre pentru fiecare enunț. Mediul de colectare a datelor a fost un birou, iar achiziția datelor s-a facut pe o perioadă de 3 luni pentru a lua în considerare variațiile naturale din vocea unei persoane pe parcursul timpului.

Pentru experimentele de verificare cu bazele TIMIT și NTIMIT, cei 168 de vorbitori din partea de test au fost folosiți. Ca și la experimentul de identificare, modelele vorbitorilor bazate pe mixturi Gaussiene cu 32 componente au fost antrenate folosind 8 enunțuri cu o durată totală de aproximativ 24 secunde. Cele doua enunțuri rămase de aproximativ 3 secunde fiecare au fost folosite individual ca pentru testare. Experimentele au fost efectuate cu fiecare vorbitor în rolul de pretendent, în timp ce vorbitorii rămași (excluzând vorbitorii de fundal ai pretendentului) au avut rolul de impostori. Ca la experimentele de identificare, acuratețea a fost aproape maximă pentru TIMIT și semnificativ mai slabă pentru NTIMIT. Comparând experimentele cu și fără vorbitori de fundal, s-a constatat că includerea vorbitorilor nesimilari a îmbunătățit semnificativ performanța prin o modelare mai buna a populației de impostori.

Pentru Switchboard, testele au avut la bază 24 de pretendenți, fiecare reprezentat de o mixtură Gaussiană cu 64 de componente prin utilizarea a trei minute de vorbire extrase în mod egal din trei conversații. Un total de 97 enunțuri ale pretendenților cuo durată medie de 16 secunde au fost folosite. Pretendenții aveau între unul și șase teste adevărate cu o medie de 4. Un set separat de 428 de enunțuri cu o medie de 16 secunde de la 210 vorbitori a fost folosit pentru testul pentru impostori.

Prin analiza rezultatelor, s-a constatat că se obține o eroare mai mică prin utilizarea unui set de vorbitori de fundal care să conțină atât bărbați, cât și femei.

Pentru YOHO, fiecare vorbitor a fost modelat de o mixtură Gaussiană cu 64 componente, antrenată pe baza a patru sesiuni de înrolare. Fiecare vorbitor avea 10 sesiuni de verificare a câte 4 enunțuri fiecare. Experimentul a constat în a folosi fiecare vorbitor ca și pretendent, în timp ce vorbitori rămași (excluzând vorbitorii de fundal ai vorbitorului) au avut rolul de impostori. S-au obținut o rată de falsă acceptare de 0,01%, iar de falsă rejecție de 0,1%. Se poate observa că putem obține rate ale erorilor foarte mici datorită calității ridicate a semnalului audio și a constângerilor de vocabular. Constrângerile implică faptul că modelul unui vorbitor are nevoie de un spațiu acustic constrâns și astfel permițând unui model dependent de text să fie totuși eficient pentru antrenare și testare folosind date independente de text.

Aceeași performanță ridicată se obține și la identificare folosind aceleași date: acurateț de 99,7% pentru bărbați, 97,8% pentru femei și 99,3% pentru seturi mixte. [12][13][14]

Capitolul II

Java

2.1 Prezentare generală

Java este un limbaj de programare pe calculator concurent, bazat pe clase și obiect-orientat și conceput special ca să aibă cât mai puține dependențe de implementare posibil. A fost gândit astfel încât dezvoltatorii de aplicațiisă poată scrie o singură dată și să ruleze oriunde. Codul care rulează pe o pltformă nu trebuie recompilat ca să ruleze pe o alta. Aplicațiile Java sunt de obicei compilate în bytecode care poate rula pe orice mașină virtuală Java (Java virtual machine – JVM), indiferent de arhitectura calculatorului.

Este unul dintre cele mai populare limbaje de programare folosite, mai ales pentru aplicațiile web de tip client-server, cu peste 9 milioane de dezvoltatori [15]. Java a fost creat inițial de Jams Gosling și a ieșit pe piață în 1995 ca o componentă de bază a platformei Java de la Sun Microsystems. O mare parte a sintaxei provine de la limbajele C și C++, dar cu mai puține funții de nivel scăzut.

Java avea cinci obiective principale atunci când a fost creat:

- Să fie simplu, obiect-orientat și cunoscut.
- Să fie robust și sigur.
- Să fie independent de arhitectură și portabil.
- Să fie executat cu performanțe ridicate.
- Să fie interpretat, cu mai multe fire de execuție și dinamic.

2.2 Programarea obiect-orientată

Este un model de programare eficient utilizat în industria software pentru dezvoltarea de aplicații cu complexitate medie și mare. Permite reducerea costurilor de dezvoltare, reducerea timpilor în care aplicațiile ajung pe piață, simplifică întreținerea și extinderea programelor și permite dezvoltarea pltformelor comune pentru grupuri și aplicații. Este caracterizată de posibilități complexe de organizare li structurare a implementărilor, facilitează reutiliziarea codului și are capacitatea de a trece în mod natural de la soluții specifice la soluții generice.

Pentru a dezvolta cu succes aplicațiile software, capacitatea de a clasifica și organiza conceptele implicate de o problemă este esențială.

Un program obiect-orientat este format dintr-un grup de obiecte care cooperează în vederea atingerii unor obicetive comune. Ideea fundamentală a limbajelor obiect-orientate este aceea de a combina într-o singură entitate de program atât datele cât și funcțiile care operează asupra datelor. Enitatea poartă numele de obiect.

Funcțiile unui obiect sunt funcții membre (aparțin unei clase particulare de obiecte) și sunt modul de a accesa datele. Se mai numesc și metode. Pentru a citi datele dintr-un obiect, se apelează o funcție membru din obiect care citește și prelucrează datele și returnează valoarea acestora. Nu se pot accesa direct datele, acestea sunt ascunse deci sunt sigure și nu se pot altera accidental. Datele și

funcțiile sunt încapsulate într-o singură entitate. Dacă se dorește modificare datelor în obiect, se cunoaște exact care sunt funcțiile care interacționează cu acestea, adică funcțiile membre. Acest lucru are ca avantaj simplificarea scrierii, testării, întreținerii și folosirii resurselor reutilizabile.

Modelul obiect al unei aplicații implică patru principii importante:

- Abstractizare – este posibilitatea ca un program să ignore unele aspecte ale informației pe care o manipulează și de a se concentra asupra esențialului. Procesele, funcțiile sau metodele pot fi de asemenea abstracte, dar în acest caz este nevoie de o varietate de tehnici pentru a extinde abstractizarea.
- Încapsulare – asigură faptul că obiectele nu pot schimba starea internă a altor obiecte în mod direct, ci doar prin metode puse la dispoziție de obiectul respectiv. Doar metodele proprii ale obiectului pot accesa starea acestuia. Fiecare tip de obiect oferă o interfață pentru celelalte obiecte care specifică modul cum acele obiecte pot interacționa cu el.
- Polimorfism – abilitatea de a procesa obiectele în mod diferit, în funcție de tipul sau clasa lor. Mai exact, este abilitatea de a redefini metode pentru clasele derivate.
- Moștenirea – organizează și facilitează polimorfismul și încapsularea, permițând definirea și crearea unor clase specializate plecând de la clase deja definite, împărtășind și extinzând comportamentul lor fără să-l redefinească. Conceptul de moștenire permite construirea unor clase noi care păstrează caracteristicile și comportamentul de la una sau mai multe clase deja existente, numite clase de bază. Aceste noi clase pot redefini date și funcții sau pot adăuga unele noi. O clasă moștenitoare a unei sau ai multor clase de bază se numește clasă derivată.

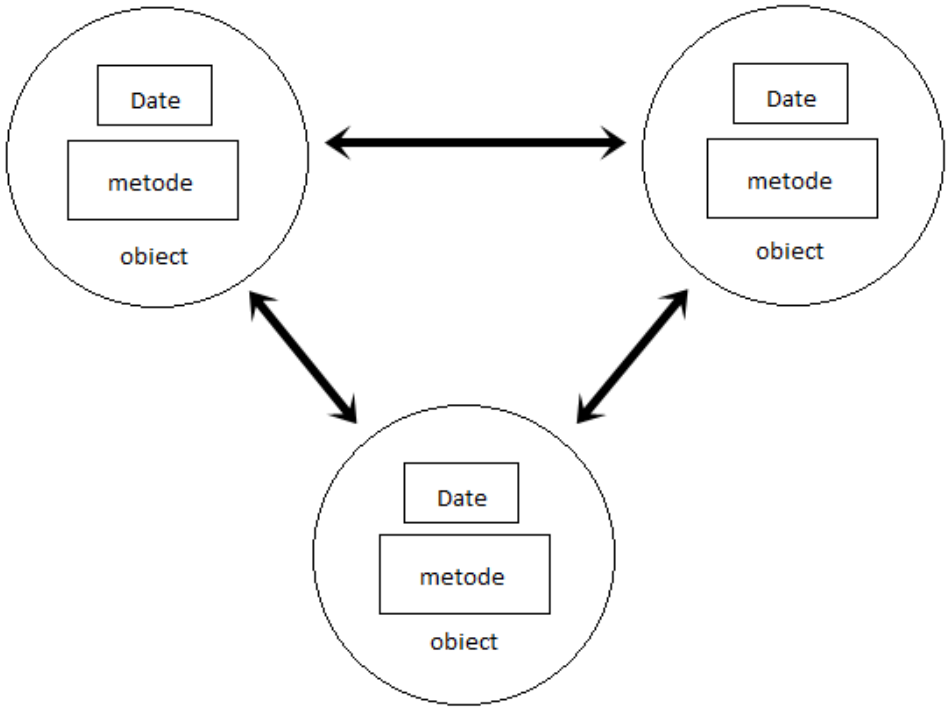

Figura 2.1 Reprezentare grafică a modelului de programare obiect-orientat

Un tip de date într-un limbaj de programare este o reprezentare a unui concept. O clasă este un tip definit de utilizator care reunește date și funcții. Acest tip poate fi folosit pentru a declara obiecte de acest tip, iar un obiect este instanța unei clase.

O clasă bine definită permite încapsularea, adică protejarea accesului la starea unui obiect. Conform acestui principiu, asupra unui obiect se poate acționa numai prin funcțiile pe care acesta le pune la dispoziție în interfață și care sunt tip `public`. Protecția datelor se face prin identificatorii `private` sau `protected`.

O dată membră a unei clase poate fi declarată de tip `static` în declarație. Asta înseamnă ca va exista o singură copie a unei date de tip static care nu aparține niciunuia dintre obiectele clasei, dar este comună tuturor. Cea mai frecventă utilizarea a datelor statice este de a asigura accesul la o variabilă comună tuturor instanțelor și pot înlocui variabilele globale.

O funcție de tip `static` se declară în interiorul clasei și se definește în interiorul clasei sau în afara acesteia, folosind operatorul de rezoluție. O funcție statică are vizibilitatea limitată la fișierul în care a fost definită și este independentă de instanțele clasei. O astfel de funcție nu poate avea acces decât la datele statice ale clasei și la datele și funcțiile globale ale programului.

Constructorii sunt apelați de fiecare dată când este creat un obiect al unei clase care are constructor, definit sau generator de compilator, cu scopul de a inițializa starea acestuia. El crează structura de bază a obiectului. Destructorul este metoda apelată în mod automat când un obiect își incheie existența pentru a elibera resursele. Daca destructorul sau constructorul sunt declarate private, clasa nu se poate instanția.

Obiectele se pot aloca în memoria liberă folosind operatorul `new`. La alocarea memoriei pentru un singur obiect se pot transmite argumente care sunt folosite pentru inițializarea obiectului, prin apelul acelei funcții constructor a clasei care se potrivește cel mai bine cu argumentele de apel. Eliberarea memoriei ocupate de un obiect se realizează prin operatorul detele, care apelează implicit destructorul clasei.

Derivarea permite definirea într-un mod simplu, eficient și flexibil a unor clase noi prin adăugarea unor funcționalități claselor deja existente. Clasele derivate exprimă relații ierarhie între conceptele pe care acestea le reprezintă și asigură o interfață comună pentru mai multe clase diferite. O clasă de bază virtuală este moștenită o singură dată și creează o singură copie în clasa derivată și reprezintă o soluție pentru eliminarea ambiguităților în moștenirile multiple.

Redefinirea unei funcții virtuale într-o clasă derivată dominî definiția funcției din clasa de bază. Mecanismul de virtualitate asigură selecția funcției redefinite în clasa derivată numai la apelul funcției pentru un obiect referit prinr-un pointer [26]

2.3 Platforma Java

Una din caracteristicile Java este portabilitatea, ceea ce înseamnă că programele scrise în limbajul Java trebuie să ruleze asemănător pe orice sistem, indiferent de hardware sau sistem de operare. Acest lucru se realizează prin compilarea programelor într-o reprezentare numită Java bytecode în loc să fie transformate în cod-mașină specific platformei pe care rulează. Bytecode-ul este asemănător codului mașină, dar conceput pentru a fi interpretat de o mașină virtuală specifică sistemului care găzduiește procesul.

Librăriile standardizate oferă o cale de a accesa trăsăturile specifice gazdei precum elemente grafice, fire de execuție multiple și rețelistică.

Deținătorul curent al implementării oficiale a platformei Java SE este Oracle, în urma achiziției Sun Microsystems în 2010. Această implementare se bazează pe cea originală a firmei Sun. Este disponibilă pentru Mac Os, Windows și Solaris.

Unele platforme oferă suport hardware direct pentru Java. Există microcontrolere care rulează Java direct pe hardware, nu pe software și procesoarele pe bază de ARM pot avea suport hardware pentru execuția de bytecode Java prin Jazelle, o stare de execuție concepută special pentru acest scop.

Java are de asemenea un sistem automat de administrare a memoriei. Programatorul determină când sunt create obiectele și Java este responsabil de recuperarea memoriei când obiectele nu mai sunt folosite. Odată ce nu mai există referințe la un obiect, memoria la care nu se mai poate ajunge poate fi eliberată automat. Una din ideile din spatele modelului Java de administrare a memoriei este de a scuti programatorii de a face manual această administrare. În unele limbaje, memoria necesară pentru a crea obiecte este alocată implicit în stivă sau alocată și dealocată explicit din spațiul liber. În al doilea caz, responsabilitatea administrării memoriei revine programatorului. Dacă programul nu dealocă un obiect, apare o scurgere de memorie (memory leak). Dacă se încearcă accesul sau dealocarea memoriei care a fost deja dealocată, atunci rezultatul este greu de prevăzut, dar cel mai probabil programul devine instabil și se poate prăbuși. Sistemul de administrare nu previne scurgerile logice de memorie, adică atunci când un obiect este declarat, dar nu și folosit.

Sintaxa Java provine în mare parte de la C++. Tot codul este scris în interiorul unei clase și fiecare element e reprezentat de câte un obiect, cu excepția primitivelor (numere întregi, numere în virgulă mobilă, valori logice și caractere). Spre deosebire de C++, Java nu suportă supraîmcarcarea operatorilor și nici moștenirile multiple pentru clase. Acest fapt simplifică limbajul și ajută la prevenirea unor eventuale erori.

Java are câteva clase speciale:

- Applet – programe încorporate în alte aplicații, de obicei o pagină web; utilizatorul folosește applet-ul într-un proces separat de cel al browserului web; poate să fie deschis într-un cadru al paginii web, într-o nouă fereastră sau într-un utilitar pentru testarea applet-urilor
- Servlet – mecanism pentru exinderea funcționalității unui server web și pentru accesarea sistemelor deja existente; sunt componente ale părții de server și generază răspunsuri, de obicei de tip pagini HTML, la cereri de la clienți.
- Pagini JavaServer (JSP) – componente ale părții de server care generează răspunsuri, de obicei pagini HTML, la cereri de tip HTTP de la clienți; paginile JavaServer încorporează cod Java într-o pagină HTML prin delimitatori speciali; o JSP este compilată într-un servlet prima dată când e accesată, apoi servlet-ul generează răspunsuri.
- Aplicații Swing – Swing este o librărie pentru interfață grafică a platformei Java; se poate specifica un aspect diferit prin sistemul “pluggable look and feel”.
- Generice – până la introducerea genericelor, fiecare declarație a unei variabile trebuia să fie de un anumit tip, ceea ce reprezenta o problemă pentru clasele container deoarece nu exista vreo metodă ușoară de a crea un container care să accepte doar anumite tipuri de obiecte; containerul poate fie să opereze pe toate subtipurile unei clase sau interfețe, sau trebuie creată câte o clasă container pentru

fiecare clasă conținută; genericele permit unei metode să opereze asupra obiectelor de tipuri variate.

2.4 Librării

- Librăriile de bază
  - Funcții de nivel scăzut, GUI, Integration, Deployment, Tools
  - Librăriile de colecții care implementează structuri de date ca liste, dicționare, arbori, seturi, cozi și stive
  - Procesare de XML (Parsing, Transformare, Validare)
  - Securitate
  - Librării de internaționalizare și localizare
- Librăriile de integrare, care permit celui care scrie aplicația să comunice cu sistemele externe
  - Java Database Connectivity (JDBC) pentru accesul bazelor de date
  - Java Naming and Directory Interface (JNDI) pentru căutare li descoperire
  - RMI și CORBA pentru dezvoltarea aplicațiilor distribuite
  - JMX pentru administrarea și monitorizarea aplicațiilor
- Librăriile de interfațare cu utilizatorul
  - AWT
  - Swing
- Implementare specifică platformei a mașinii virtuale Java cu care se execută bytecode
- Plugins, pentru a permite applet-urilor să ruleze în browsere web
- Java Web Start, care permite aplicațiilor Java să fie distribuite eficient utilizatorilor finali prin intermediul internetului
- Licență și documentație

Capitolul III

Construcția sistemului bazat pe LIUM

3.1 Prezentare generală a LIUM

LIUM_SpkDiarization este un software dedicat diarizării, adică partiționarea unor înregistrări audio în segmente în funcție de identitatea vorbitorului. Acestă împărțire se face fară să fie nevoie de alte informații in prealabil. Nu este nevoie să se cunoască numarul de vorbitori, nici identitatea lor și nu necesită nici mostre ale vocii lor. Etichetele reprezentând numele vorbitorilor sunt generate automat, fără să corespundă indentității reale a vorbitorului. O abordare folosită în multe cazuri este de a detecta segmente audio omogene, adică să conțină vocea unui singur vorbitor. Segmentele generate sunt apoi adunate în grupuri sau clustere, un grup conținând un singur vorbitor.

Metodele principale folosite recent se bazează pe grupare ierarhică. Metodele diferă în alegerea distanței de îmbinare, criteriului de oprire și tipului de model de grup folosit (mono-Gaussian sau mixturi Gaussiene). Arhitectura sistemului este parțial dată de natura documentele cu care va funcționa sistemul.

Mai multe utilitare open-source sunt disponibile pe internet, precum ALIZE Speaker Diarization, Audioseg sau ShoUT [9].

LIUM_SpkDiarization poate fi folosit în mai multe moduri, având ca țintă aplicații diverse. În configurația de bază, este orientat către diarizarea știrilor. Utilitarul oferă unelte de bază, precum segmentare, generatoare de grupuri, decodor si antrenor de modele. Îmbinarea acestor elemente este o modalitate ușoară de a dezvolta un sistem de diarizare specifică, dar poate fi utilizat și pentru sarcini de identificare și verificare.

```
#!/bin/bash show="ubm"

# Input segmentation file, %s will be substituted with $show
seg="./%s.seg"

# Where is the mfcc, %s will be substituted with the name of the segment show fMask="./m
# The MFCC vector description, here it corresponds to 12 MFCC + Energy
# spro4=the mfcc was computed by SPro4 tools
# 1:1:0:0:0:0: 1 = present, 0 not present.
# order : static, E, delta, delta E, delta delta delta delta E
# 13: total size of a feature vector in the mfcc file
# 1:0:0:1 CMS by cluster fDesc="spro4,1:1:0:0:0:0,13,1:0:0:1"

# The GMM used to initialize EM, %s will be substituted with $show
gmmInit="./%s.init.gmms" # The output GMM, %s will be substituted with $show
gmm="./%s.gmms"

# Initialize the UBM, ie a GMM with 8 diagonal Gaussian components
/usr/bin/java -Xmx1024m -cp ./LIUM_SpkDiarization.jar fr.lium.spkDiarization.programs.MT
--sInputMask=$seg --fInputMask=$fMask --fInputDesc=$fDesc --kind=DIAG \
--nbComp=8 --emInitMethod=split_all --emCtrl=1,5,0.05 --tOutputMask=$gmmInit $show

# Train the UBM via EM
/usr/bin/java -Xmx1024m -cp ./LIUM_SpkDiarization.jar fr.lium.spkDiarization.programs.MT
--sInputMask=$seg --fInputMask= $fMask --emCtrl=1,20,0.01 --fInputDesc=$fDesc \
--tInputMask=$gmmInit --tOutputMask=$gmm $show
```

Figura 3.1 Exemplu de script în bash pentru folosirea LIUM

3.2 Construcția sistemului

În forma sa inițială, LIUM se putea utiliza prin intermediul unor script-uri in bash și era nevoie de execuția individuala a mai multor script-uri pentru un singur proces. Testele inițiale de identificare de vorbitor au fost realizate pe o bază de date în limba daneză.

Primul pas a fost adaptarea script-urilor pentru a efectua teste pe o nouă bază de date a cărei enunțuri sunt o serie de cifre rostite în limba română (roDigits). Aceasta este o bază mai largă, conținând 69 vorbitori cu câte 100 enunțuri fiecare.

Adaptarea presupune în primul rând modificarea script-urilor care generează fișierele de tip ".seg" și ".mfcc". Fișierele de tip .mfcc sunt cele care conțin vectorii de trăsături. Atât pentru antrenare cât și pentru testare LIUM nu folosește fișierele audio, ci doar setul de vectori de trăsături extras din fiecare din clip-urile audio. Aceste trăsături, în cazul nostru coeficienți mel-cepstrali sau MFCC, trebuie calculate în prealabil și LIUM folosește un script pe bază de SphinxBase pentru a face acest lucru.

Fișierele de tip .seg conțin o singură linie și sunt de forma „[file_id] [channel] [start_frame] [end_frame] [gender] [bandwidth] [environment] [speaker_id]", iar descrierea fiecărui element este următoarea:

- [file_id] – denumirea fișierului cu trăsături
- [channel] – numarul canalului de comunicație
- [start_frame] – numărul ferestrei de început
- [end_frame] – numărul ultimei ferestre
- [gender] – sexul vorbitorului: M – masculin, F – feminin, U - necunoscut
- [bandwidth] – lărgimea de bandă
- [environment] – mediul în care s-a făcut înregistrarea (liniște, zgomot, necunoscut etc.)
- [speaker_id] – identitatea vorbitorului, în funcție de care este fișierul este grupat în cluster-ul cu același nume.

După execuția testelor pe noua bază de date, am trecut la investigarea codului LIUM pentru a putea trece tot procesul în Java.

Antrenarea sistemului se petrece în mai multe etape.

1. initEM - inițializează un model universal de fundal (universal background model - UBM) care este un model bazat pe trăsăturile tuturor vorbitorilor sistemului; se pornește de la o singură Gaussiană care este apoi împărțită și antrenată iterativ până se atinge numarul de componente țintă
2. trainEM – antrenează GMM-ul prin metoda EM (estimatiom maximization); după 1 până la 20 iterații, algorimul se oprește daca s-a diferența de probabilitate între doua iterații este sub un anumit prag.
3. initMAP – inițializează modelul propriu al fiecărui vorbitor prin copierea UBM-ului obținut la punctul anterior.
4. trainMAP – adaptează modelul universal prin metoda MAP (maximum a posteriori).

O problemă semnificativă a fost antrenarea UBM. Aceasta se realiza prin concatenarea tuturor fișierelor audio într-un fișier unic și extragerea parametrilor din acesta. Acestă abordare era foarte ineficientă deoarece presupune dublarea dimensiunii bazei de date și creșterea semnificată a timpului de lucru. Am rezolvat acest aspect prin crearea unor fișiere .seg separate, care sunt asemănătoare cu fișierele .seg originale, dar identitatea vorbitorului a fost înlocuită cu "spk_UBM". Asta înseamnă că ele vor fi grupate într-un singur cluster și interpretate ca provenind de la un singur vorbitor, astfel putând să antrenăm UBM-ul cu o listă de clipuri audio folosind parametrii deja extrași. Fișierele acestea se crează foarte rapid și ocupă foarte puțin spațiu, oferind o alternativă mult mai bună față de concatenarea tuturor fișierelor.

După realizarea acestor metode, următoarea etapă a fost crearea unei clase pentru generarea listelor de fișiere, atât pentru antrenare cât și pentru testare. Fiecare vorbitor având 100 fișiere, această clasă permite alegerea indicilor fișierelor care vor fi folosite pentru fiecare caz în parte.

O altă eroare întâlnită a fost „Comparison method violates it's general contract". Aceasta provenea de la schimbări apărute în Java la trecerea de la JDK 1.6 la JDK 1.7. Pentru a rezolva această problemă, am avut opțiunea de a reveni la JDK 1.6, dar soluția pe care am ales-o a fost să utilizez proprietatea `System.setProperty("java.util.Arrays.useLegacyMergeSort", "true")` care îmi permite să folosesc Merge Sort-ul fără să fie nevoie să fac downgrade versiunii de Java.

Având la dispoziție clasele nou create pentru antrenare și decodare, am realizat teste pe baza de date în limba daneză folosind UBM-ul realizat dintr-un singur fișier, cel realizat folosind o listă de fișiere și cel antrenat inițial care a venit stocat cu baza de date. Rezultatele obținute au fost foarte slabe, detaliate în subcapitolulde experimente a acestui capitol. Am refăcut testele, încercând cu parametrii diferiți, de exemplu numărul de componente sau numărul de iterații, dar rezultatele s-au schimbat nesemnificativ. În urma unei investigații de durată, am aflat că problema provenea de la un nivel de bază. Realizând din nou testele cu parametrii cu care a venit baza de date, rezultatele au fost mult mai bune. Am identificat problema ca fiind endianess-ul fișierelor mfcc, diferit pe sistemul cu care am lucrat față de cel pe care au fost extrași parametrii. Cercetând codul LIUM, am descoperit variabila de tip boolean `swap` a clasei FeatureSet care poate fi setată pentru a indica direcția de citire a fișierelor cu trăsături, rezolvând problema endianess-ului.

Următorul pas a fost gasirea unui mod de a extrage scorurile pentru fiecare fișier decodat, atât scorul maxim, în funcție de care se ia decizia de verificare, cât și scorurile pentru fiecare vorbitor, în cazul identificarii, pentru eventuala stabilire a unui prag. Acestea sunt scoase în timpul execuției metodei `make` a clasei MScore din cadrul LIUM și sunt scrise într-un fișier separat. Ulterior am realizat clasa ResultsInterpretation care să separe rezultatele în 3 categorii: rezultate corecte pentru vorbitori modelați de sistem, rezultate incorecte pentru vorbitori modelați de sistem și rezultate pentru vorbitorii cu care sistemul nu a fost antrenat. Dacă nu s-a stabilit un prag de decizie, atunci această ultimă categorie conține doar rezultate incorecte deoarece sistemul nu știe că vorbitorul nu face parte din model și atribuie fișierul vorbitorului cu cel mai mare scor. Deși poate părea în plus, această categorie este importantă pentru a vedea diferențele între scorurile obținute de vorbitori modelați și nemodelați.

În continuare am realizat o clasă cu ajutorul căreia parametrii MFCC să poată fi extrași în timpul procesului de antrenare sau testare și să nu fie nevoie ca ei să existe deja la rularea programului. Am realizat acest lucru prin folosirea unui FrontEnd oferit de CMUSphinx, care este un program de recunoaștere a vorbirii continue independentă de vorbitor și cu vocabular mare, dar în același timp este și o colecție de utilitare open-source care permite dezvoltarea de aplicații bazate pe recunoștere de vorbire sau vorbitor. Cu ajutorul acestui FrontEnd, fișierul audio trece printr-un Preemphasizer, un filtru trece sus care compensează pentru atenuarea semnalului audio (de obicei 20dB/dec), apoi este împărțit în ferestre, se aplică transformata Fourier rapidă, se trece semnalul prin bancul de filtre Mel și în final se aplică transformata cosinus discretă.

Pasul final a fost încorporarea acestor clase și metode într-o aplicație de sine stătătoare care să comunice prin obiecte de tip XML și să realizeze automat atât antrenarea sistemului cât și decodarea fișierelor.

3.3 Experimente

Primele testes au fost făcute pe baza de date în limba daneză care conține 23 vorbitori cu câte 2 enunțuri fiecare. Am realizat 4 scenarii de test cu această bază de date:

- Proiect 1
  - folosește fișierele cu modelele pentru UBM si GMM deja existente
  - UBM generat cu un singur fișier format prin concatenarea tuturor fișierelor audio
  - JDK 1.7 + System.setProperty("java.util.Arrays.useLegacyMergeSort", "true")
- Proiect 2
  - folosește fișierele cu modelele pentru UBM si GMM generate local
  - UBM generat cu un singur fișier format prin concatenarea tuturor fișierelor audio
  - JDK 1.7 + System.setProperty("java.util.Arrays.useLegacyMergeSort", "true")
- Proiect 3
  - folosește fișierele cu modelele pentru UBM si GMM generate local
  - UBM generat cu o listă de fișiere
  - JDK 1.7 + System.setProperty("java.util.Arrays.useLegacyMergeSort", "true")
- Proiect 4
  - folosește fișierele cu modelele pentru UBM si GMM generate local
  - UBM generat cu o listă de fișiere
  - JDK 1.6

Pentru toate proiectele, rezultatele au fost foarte slabe, având o rată a erorii de peste 90%. După identificarea problemei cu fișierele .mfcc, am rulat din nou testele. Rata erorii a scăzut la aproximativ 10% pentru toate proiectele. Am ales să rulez viitoarele teste cu scenariul cel mai eficient, cel al proiectului 3. Rezultatele au fost aproape identice, indicând faptul că modelele pentru fiecare vorbitor și universal sunt generate corect, că se poate utiliza fără erori JDK 1.7 împreună cu proprietatea indicată și UBM-ul se poate genera mult mai ușor și mai rapid folosind o listă cu fișiere și fișiere .seg modificate față de varianta concatenării tuturor fișierelor audio.

În urma acestor experimente, am creat Proiectul 5 care are condițiile inițiale identice cu cele ale proiectului 3, diferența fiind baza de date folosită. Am utilizat roDigits, o bază de date mult mai largă cu un total de 6900 de fișiere audio față de 46 cât avea cea în limba daneză. În cadrul acestui proiect, am testat rezultatele identificării pentru un număr diferit de fișiere de antrenare. Fiecare vorbitor având câte 100 fișiere audio, am generat modele pentru vorbitori folosind câte 10, 20, 40, 60 și 80 fișiere, restul până la 100 fiind utilizate la testare. Din 69 vorbitori, s-au folosit doar 50 la antrenare. Evaluarea s-a facut cu toți vorbitorii bazei de date. În tabelul 3.1 sunt prezentate rezultatele obținute în condițiile Proiectului 5.

Număr de fișiere folosite la antrenare:	Vorbitori folosiți la decodare:	Rezultate (% corect):
80	Toți vorbitorii	56.15
60	Toți vorbitorii	53
40	Toți vorbitorii	55.26
20	Toți vorbitorii	53.96
10	Toți vorbitorii	43.83
80	Doar vorbitorii folosiți la antrenare	77.5
60	Doar vorbitorii folosiți la antrenare	73.15
40	Doar vorbitorii folosiți la antrenare	76.26
20	Doar vorbitorii folosiți la antrenare	74.47
10	Doar vorbitorii folosiți la antrenare	60.48

Tabelul 3.1 Rezultatele decodării pentru Proiectul 5

Cand s-au folosit toți vorbitorii pentru evaluare am obținut rezultate semnificativ mai slabe față de cazul în care am folosit doar vorbitorii modelați. În lipsa unui prag de decizie, sistemul consideră că se lucrează cu un set închis, asta însemnând că vorbitorul se află neapărat printre modelele stocate. Astfel, vorbitorului care nu face parte din sistem îi este atribuită identitatea vorbitorului modelat pentru care s-a obținut cel mai mare scor.

În cazul în care s-au folosit doar vorbitorii cu care a fost antrenat sistemul, rezultatele sunt mai bune. Se poate observa o diferență destul de mare la trecerea de la 10 la 20 fișiere folosite la antrenare, dar după aceast prag de 20 fișiere schimbările sunt nesimnificative. În cadrul aplicației, am ales să lucrez cu 10 fișiere de antrenare pentru fiecare vorbitor, un compromis între eficiență și timpul necesar atât pentru antrenare cât și pentru înrolare.

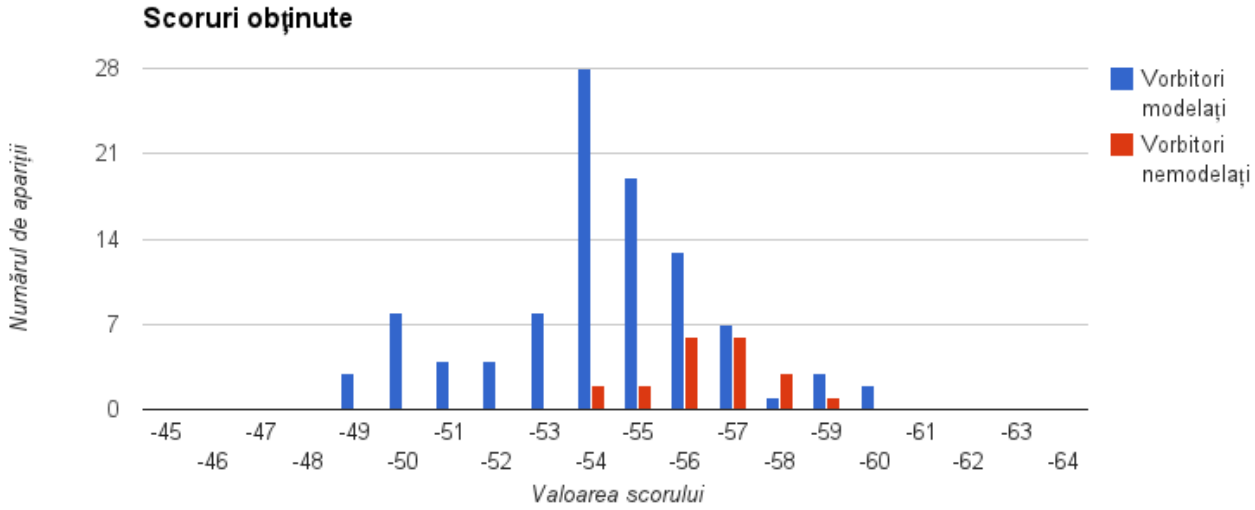

Figura 3.2 Scoruri obținute pentru 6 vorbitori, 5 modelați si 1 nemodelat

În figura 3.2 se poate observa plaja de valori în care se află scorurile și numărul de apariții al fiecărui scor exemplificate pentru 6 vorbitori. Aceste scoruri sunt importante pentru a vedea diferențele între vorbitori care aparțin sistemului și eventuali impostori pentru a putea stabili un prag în funcție de care să se poată lua decizia de verificare sau identificare.

Capitolul IV

Aplicația

Partea practică a proiectului este o aplicație client-serve realizată integral în Java. După pornirea serverului și încărcarea paramterilor inițiali, un utilizator se poate conecta folosind clientul. În momentul în care un nou client se conectează, serverul pornește un client peer care se ocupă de comunicarea cu acesta. Utilizatrul are de ales dintre înrolareîn sistem, verificare sau identificare.

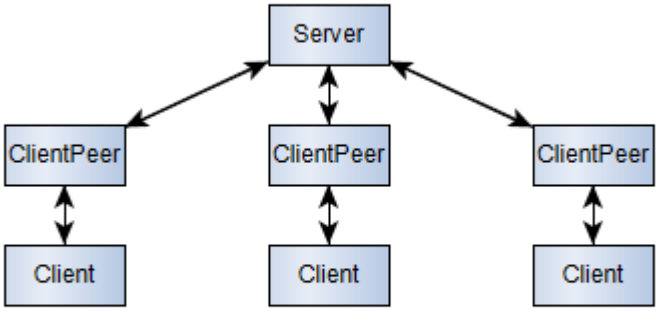

Figura 4.1 Structura generală a aplicației

Structura aplicației poate fi împărțită în două categorii: client (LiumSpeakerRecognitionClient) și server, iar partea de server poate fi și ea împărțită în server propriu-zis (LiumSpeakerRecognitionServer) și client peer (LiumSpeakerRecognitionClientPeer). Aceste clase principale folosesc în funcționarea lor multe clase auxiliare create pentru această aplicație, printre cele mai importante fiind Trainer, Decoder, FileListBuilder, XMLBuilder, DataWriter, Speaker și SpeechUtterance.

4.1 LiumSpeakerRecognitionServer

Variabile globale:

- serverSocket – variabilă de tip ServerSocket reprezentând portul cu care va rula aplicația; preluat din argumente funcției main.
- UBMFilePath – variabilă de tip String, reține calea catre directorul ce conține UBM; citită din fișierul de proprietăți.
- speakerGMMsPath – variabilă de tip String, reține calea catre directorul ce conține modelul pe bază de GMM al fiecărui vorbitor; citită din fișierul de proprietăți.
- databaseFolderPath – variabilă de tip String, reține calea catre directorul ce conține fișierele cu care va lucra aplicația, citită din fișierul de proprietăți.

- resultsFolderPath – variabilă de tip String, reține calea către directorul unde vor fi stocate rezultatele decodărilor.
- speakerListPath – variabilă de tip String, reține calea către fișierul ce conține lista cu vorbitori; preluată din fișierul de proprietăți.
- speakerList – variabilă de tip List<Speaker>, conține lista obiectelor de tip Speaker reprezentând vorbitorii înrolați.
- nextId – variabilă de tip int, conține valoarea ID-ului ce va fi atribuit următorului utilizator ce se înregistrează și reprezintă în același timp și numărul de vorbitori înrolați.

Metode:

- public LiumSpeakerRecognitionServer(int _port) – constructorul clasei ce primește ca argument numărul portului și il atribuie variabile serverSocket.
- public String getDatabaseFolderPath() – returnează calea către baza de date.
- public String getUBMFilePath() – returnează calea către directorul ce conține UBM-ul.
- public String getSpeakerGMMsFilePath() – returnează calea către directorul ce conține modelul fiecărui vorbitor.
- public String getResultsFolderPath() – returnează calea către directorul cu rezultate
- public synchronized List<Speaker> getSpeakerList() – returnează lista cu vorbitori înrolați; este o metodă sincronizată pentru a asigura faptul că nu se va citi lista de un clientPeer în timp ce altul o modifică.
- public void setNextId(int _nextId) – setează valoarea variabilei nextId.
- public int getNextId() – returnează valoarea variabilei nextId.
- public synchronized void addNewSpeaker(Speaker _speaker) – adaugă un nou vorbitor în lista celor deja existenți (speakerList), îi setează ID-ul și incrementează valoarea variabilei nextId; metoda este sincronizată pentru a ne asigura că nu se adaugă doi vorbitori cu acelși id în același timp.
- public synchronized void writeSpeakerList() – scrie pe disc lista cu utilizatori sub forma unui fișier XML.
- private void loadConfiguration(String _configFileName) – încarcă setările inițiale și lista cu vorbitori; datele necesare se găsesc într-un fișier de proprietăți dat ca argument în metoda main.
- private void waitForClients() – așteaptă conectarea clienților și pornește câte un clientPeer pentru fiecare client.
- public static void main(String _args[]) – metoda principală a clasei

Când este pornit serverul, acesta are nevoie de două argumente pentru metoda main, anume numărul portului și calea către fișierul cu proprietăți. Daca numărul de argumente este diferit, atunci va arunca o excepție de tip `NumberFormatException`, altfel se va apela constructorul care inițializează un nou obiect de tip LiumSpeakerRecognitionServer și atribuie variabilei serverSocket numărul portului. Apoi se apelează metoda loadConfiguration care citește din fișierul cu proprietăți căile către baza de date, directorul conținând UBM, directorul conținând modelele pentru fiecare vorbitor în parte, directorul unde se vor scrie rezultatele și calea către lista cu vorbitori înrolați. Se crează directoarele unde vor fi stocate datele, dacă acestea nu există. Apoi se încarcă lista cu vorbitori care se află salvată pe disc sub forma unui fișier XML care trebuie interpretat.În final, se

apelează metoda waitForClients care așteaptă conectarea clienților. De fiecare dată când un client nou se conectează, se instanțiază un obiect de tip LiumSpeakerRecognitionClientPeer cu care va discuta clientul și care primește numarul portului cât și o referință către server.

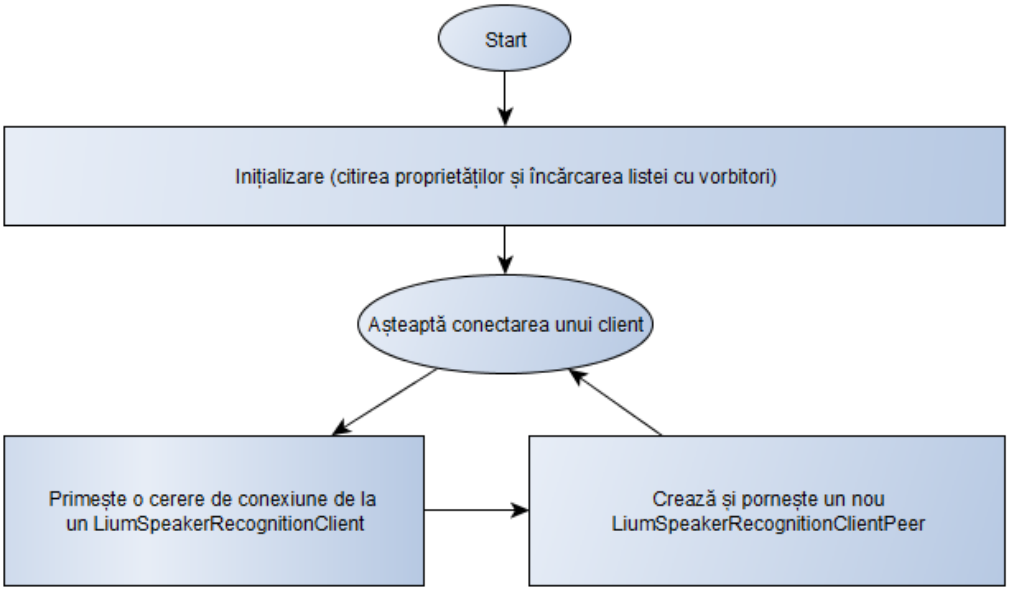

Figura 4.2 Diagramă reprezentând funcționalitatea LiumSpeakerRecognitionServer

4.2 LiumSpeakerRecognitionClientPeer

Variabile globale:

- parent – variabilă de tip LiumSpeakerRecognitionServer, conține referința către server.
- commandSocket – variabilă de tip Socket, conține numarul portului.
- outputStream – variabilă de tip XMLOutputStream prin care se transmit obiecte de tip XML către client.
- inputStream – variabilă de tip XMLInputStream prin care se primesc obiecte de tip XML de la client.
- state – variabilă de tip String inițializată cu valoarea variabilei START_STATE.
- currentSpeaker – variabilă de tip Speaker ce conține identitatea vorbitorului curent, dacă a identificată sau verificată.
- lastMaxScore – variabilă de tip double, conține valoarea scorului maxim obținut la cea mai recentă decodare.
- START_STATE - variabilă de tip String, finală, ce conține numele stării de start, starea inițială a client peer-ului.
- INPUT_STATE – variabilă de tip String, finală, ce conține numele stării de input.
- PASSWORD_VERIFICATION_STATE – variabilă de tip String, finală, ce conține numele stării de verificare a parolei
- AUTHENTICATD_STATE – variabilă de tip String, finală, ce conține numele stării autentificate

- INIT_EM_OUTPUT_FILE – variabilă de tip String, finală, ce conține numele fișierului UBM inițializat.
- TRAIN_EM_OUTPUT_FILE – variabilă de tip String, finală, ce conține numele fișierului UBM final.
- INIT_MAP_OUTPUT_PATTERN – variabilă de tip String, finală, ce conține modelul pentru numele fișierului cu modelul GMM inițializat al unui vorbitor.
- TRAIN_MAP_OUTPUT_PATTERN – variabilă de tip String, finală, ce conține modelul pentru numele fișierului cu modelul GMM final al unui vorbitor.

Metode:

- public LiumSpeakerRecognitionClientPeer(LiumSpeakerRecognitionServer _parent, Socket _socketToClient) – constructorul clasei ce primește ca argumente o referință către server și portul pe care se face comunicația cu clientul.
- public void setCurrentMaxScore(double _score) – setează valoarea variabilei lastMaxScore.
- public void run() – metoda principală a clasei

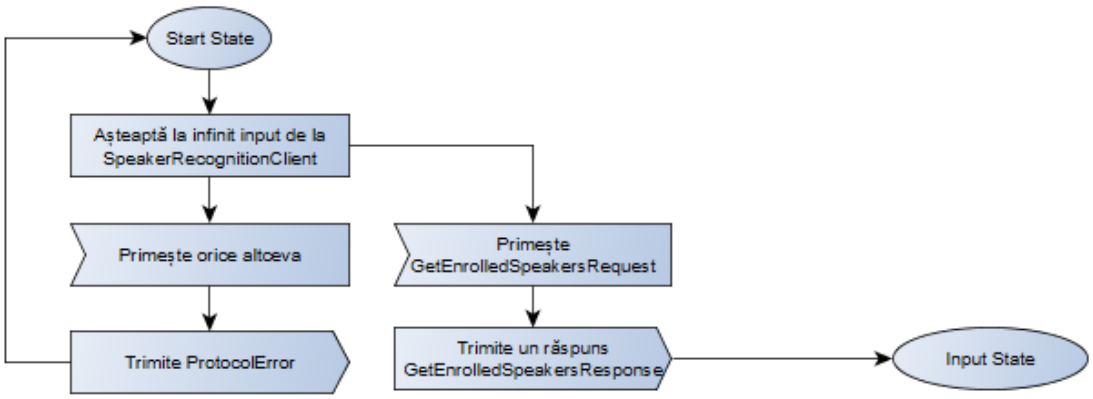

Figura 4.3 Starea de start

Client peer-ul pornește în starea de start și așteaptă informații de la client. Daca documentul nu este de tip XML, atunci se trimite o eroare de tip ”Invalid Input Stream” și se așteaptă un nou input de la client. Dacă formatul este bun, atunci se verifică daca este o cerere pentru vorbitorii înrolați. Dacă nu este o astfel de cerere, atunci se trimite o eroare de tip ”Invalid Request”, altfel se trimite lista cu vorbitori sub formă de obiect XML. Apoi se trece în starea de input.

În această stare, se așteaptă informații de la client. Dacă se primește altceva decât un document XML, atunci se trimite o eroare de tip ”Invalid Input Stream”. Dacă se primește un docunment XML, atunci se verifică ce tip este. Există trei posibilități:

- Cerere de înrolare
- Cerere de verificare
- Cerere de identificare

Dacă se primește orice altceva, se va trimite înapoi o eroare cu descrierea ”Not a valid request”.

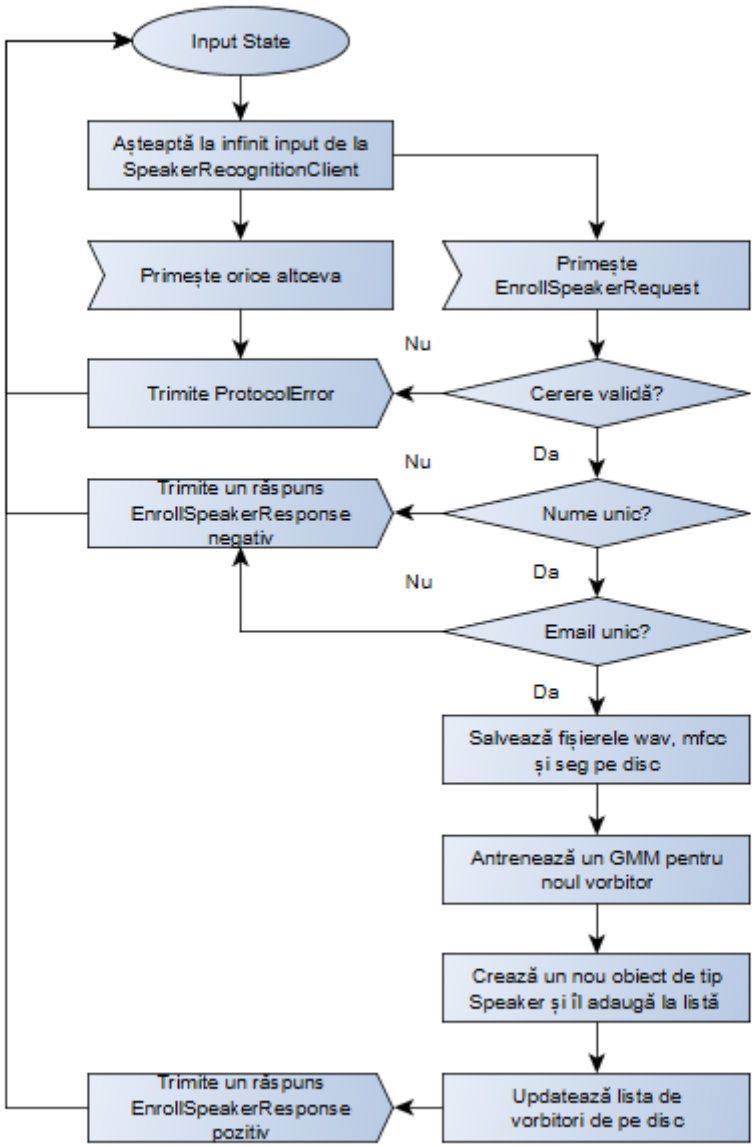

Figura 4.4 Starea de input, cazul înrolării

În cazul în care se primește o cerere de înrolare, se verifică dacă numele utilizatorului este unic și apoi se verifică daca emailul este unic. Dacă sunt deja utilizate oricare din cele două, atunci se transmite un răspuns negativ la cererea de înrolare. Dacă ele sunt unice, atunci începe procesul de înrolare. Este creat un nou obiect de tip Speaker și sunt setate numele, parola, emailul și poza pentru acest obiect, elemente care se află în cererea de înrolare. Apoi este apelată metoda addNewSpeaker a serverului care îi atribuie Speaker-ului următorul ID disponibil, incrementează acest ID și adaugă obiectul la lista de vorbitori. Apoi sunt preluate clipurile audio ce vin codate în format Base64 în cererea de înrolare. Sunt transformate într-o listă de obiecte de tip SpeechUtterance care este transmisă metodei `write` a clasei DataWriter. Această metodă decodează fișierele audio și le scrie pe disc împreună cu transcrierea lor și cu fișierele seg generate. De asemenea, în timpul acestui proces se extrag coeficienții MFCC și se scriu și aceștia pe disc. La fiecare 10 vorbitori se reantrenează modelul universal cu toți vorbitorii, apoi se generează modelul pentru vorbitorul nou

înrolat. Se scria lista modificată cu vorbitori pe disc și în final se transmite un răspuns pozitiv la cererea de înrolare, iar sistemul așteaptă un nou input de la client.

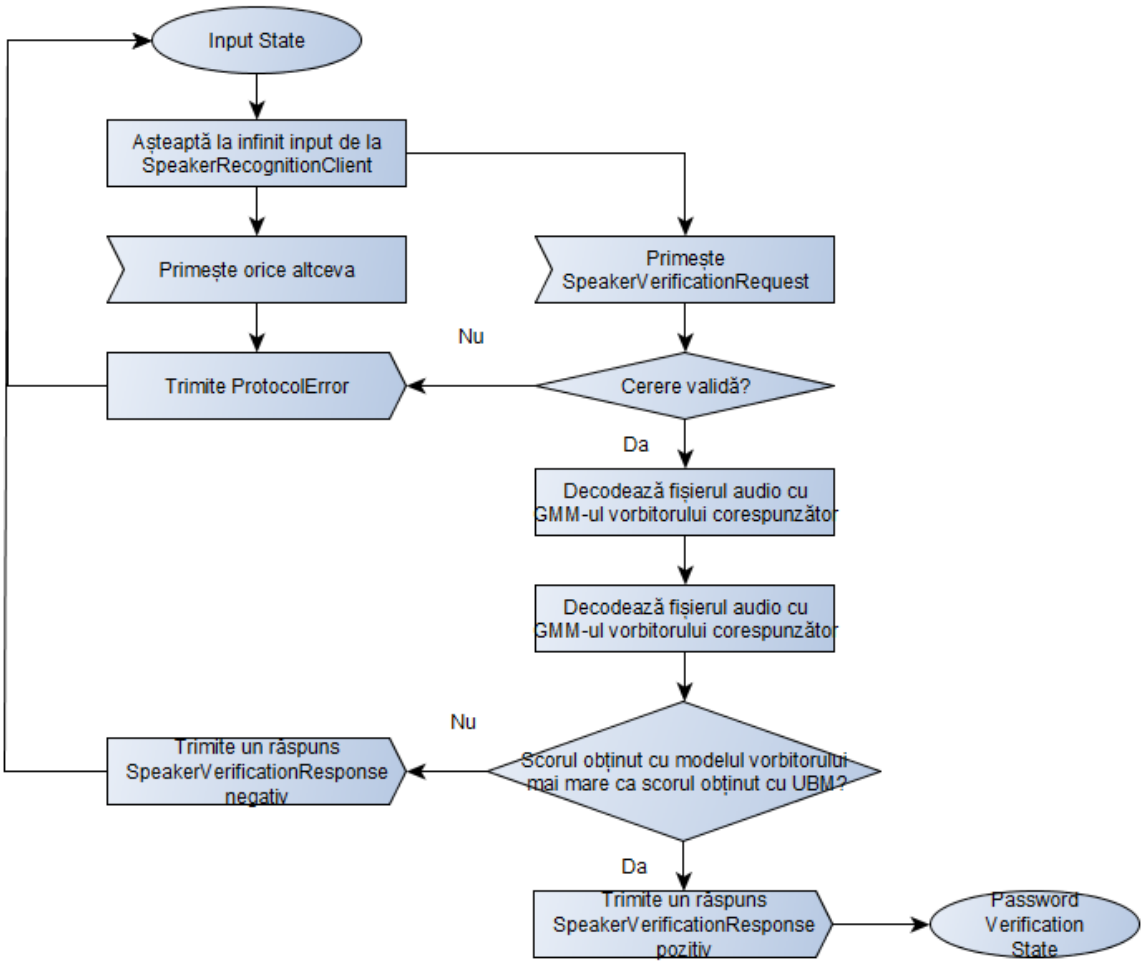

Figura 4.5 Starea de input, cazul verificării

În cazul verificării, primul pas este găsirea în lista cu vorbitori a celui care a cerut verificarea. Clientul poate doar selecta un vorbitor din lista celor deja existenți, nu introdus manual, astfel încât nu există riscul ca vorbitorul să nu se afle în listă. Apoi este extras clipul audio din cererea de verificare și este salvat pe disc împreună cu transcrierea vorbirii folosind metoda `write` a clasei DataWriter. În timpul acestui proces se generează și fișierele seg și se extrag coeficienții MFCC. Clipul audio primit este în continuare decodat de două ori: mai întâi folosind modelul universal pe post de model al vorbitorului și apoi utilizând modelul vorbitorului care a cerut verificarea. Scorurile maxime de la ambele decodări sunt reținute și comparate. Dacă vorbitorul seamănă mai mult cu modelul său decât cu modelul pentru toți vorbitorii înrolați, atunci identitatea asumată este cea corectă și se transmite un răspuns pozitiv la cererea de verificare, apoi se trece în starea de verificare a parolei. În caz contrar, utilizatorul nu a fost verificat și se transmite un răspuns negativ la cererea de înrolare și sistemul rămâne în starea de Input.

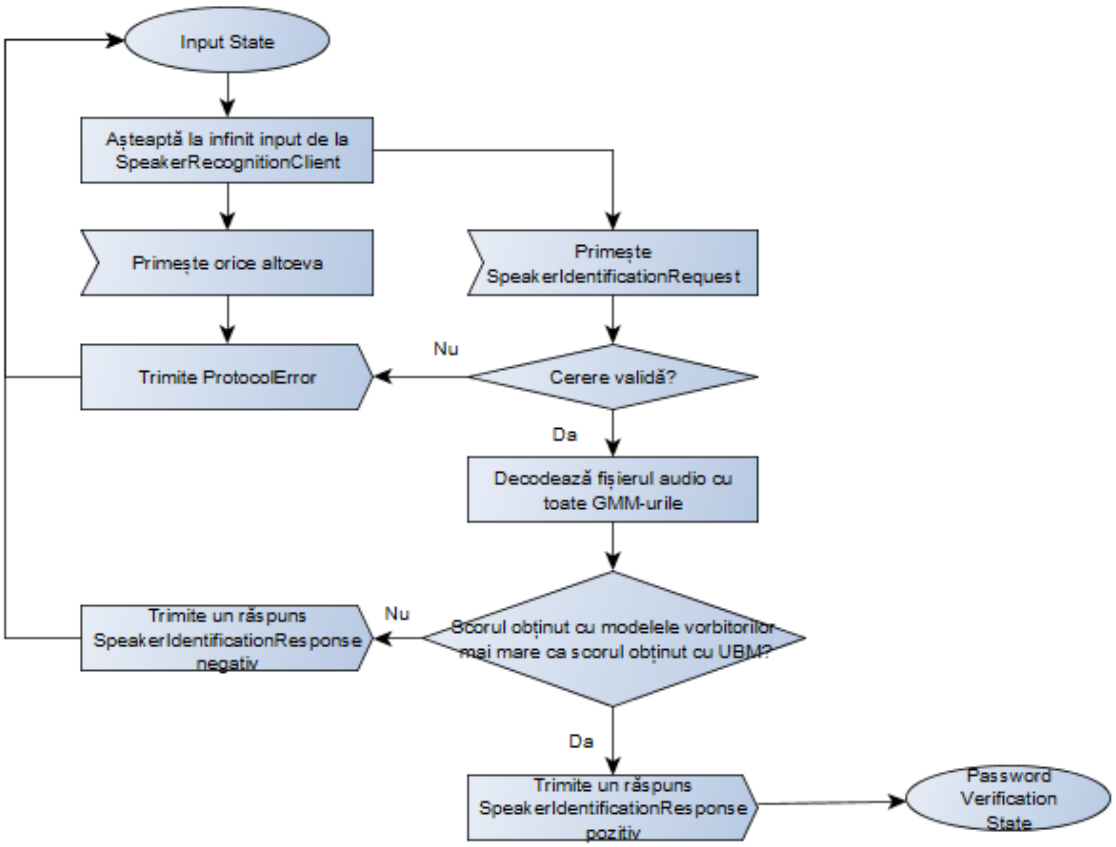

Figura 4.6 Starea de input, cazul identificarii

În cazul identificării, se așteaptă de la client o cerere validă de identificare de vorbitor. Dacă se primește orice altceva, se trimite un mesaj de eroare corespunzător. Dacă cererea este validă, atunci ea va conține un fișier audio care va fi extras, scris pe disc și i se vor genera fișierele .seg și .mfcc. Apoi se va decoda fișierul, pe rând, folosind toate modelele disponibile, inclusiv UBM-ul, și se reține scorurile maxime. Dacă la sfârșit cel mai mare scor este al UBM-ului, atunci vorbitorul care a cerut identificarea nu face parte din sistem și se trimite un răspuns negativ la cererea de identificare. Altfel, se transmite un răspuns pozitiv cu numele vorbitorului identificat.

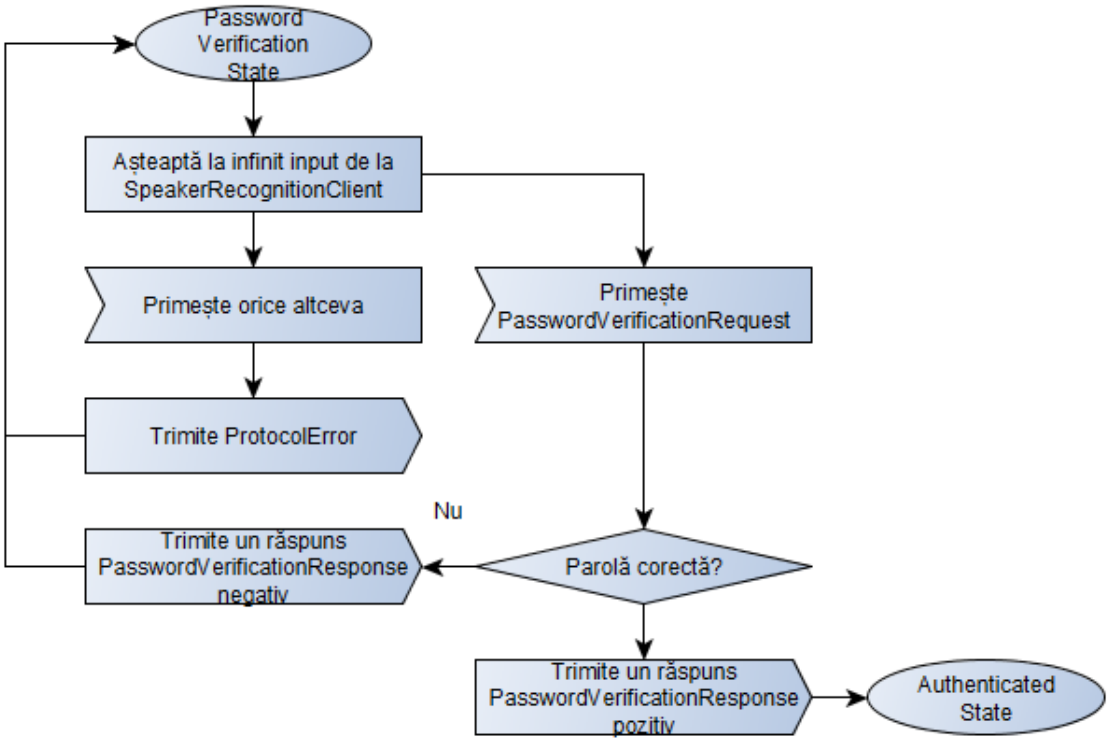

Figura 4.7 Starea de verificare a parolei

În starea de verificare a parolei, se așteaptă de la client o cerere de verificare a parolei. Dacă cererea este validă, atunci se scoate parola transmisă de client și se compară cu parola reținută pe server pentru utilizatorul curent, identificat sau verificat anterior. Dacă parola nu corespunde, atunci se transmite un răspuns negativ la cerere. Altfel, se transmite un răspuns pozitiv și se trece în starea autentificată.

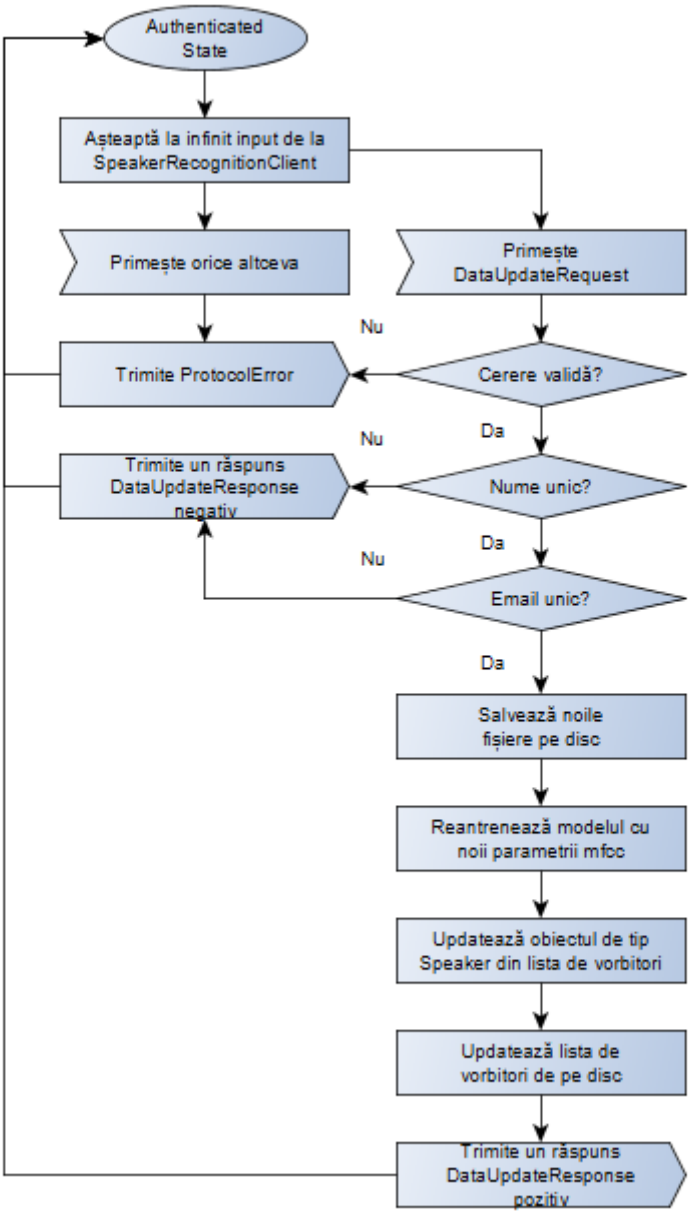

Figura 4.8 Starea autentificată

În starea autentificată, clientul are două opțiuni: să updateze informațiile și clipurile audio sau să încheie conexiunea. Dacă alege varianta de updatare a utilizatorului, atunci poate schimba numele, emailul, parola și numele. Dacă numele sau parola nu sunt unice, atunci se transmite un răspuns negativ la cererea transmisă de client. Altfel se extrag din cererea de actualizare clipurile audio codate Base64 și se scriu pe disc, cu toate fișierele aferente, iar modelul pentru vorbitor este reantrenat cu noile și vechile fișiere pentru a obține un model mai exact al vorbitorului. Obiectul de tip Speaker din lista de vorbitori este actualizat, iar noua listă este scrisă pe disc. Se trimite clientului un răspuns pozitiv la cererea de updatare a vorbitorului.

4.3 LiumSpeakerRecognitionClient

Clientul este realizat în Java folosind librăria Swing, principala metodă de a oferi o interfață grafică pentru programele în Java.

La pornirea clientului, acesta se conectează la server (care crează un nou client peer) și transmite o cerere pentru utilizatorii înrolați. După ce primește răspuns la cerere, se afișează un panou pentru calibrare, care să testeze microfonul utilizatorului. Apoi se afișează un JTabbedPane cu trei taburi: înrolare, verificare și identificare.

Panoul de înrolare conține câmpurile goale pentru nume, parolă, confirmare parolă, email, poză cu buton de cautare alături, pentru selectarea acesteia. Panoul mai conține încă doua câmpuri, unul cu numărul eunuțului și altul cu enunțul în sine, și lângă acestea un buton care să pornească și să oprească înregistrarea.

Utilizatorul completează câmpurile goale, apoi apasă pe butonul de înregistrare, citește enunțul și apasă din nou pe buton pentru a opri înregistrarea, apoi repetă ultimul pas de 10 ori. Enunțurile sunt secvențe de 12 cifre generate aleator ce se modifică dupa fiecare înregistrare. După ce toate câmpurile sunt completate și toate enunțurile înregistrare, utilizatorul apasă pe butonul de salvare a datelor. Clientul formează un document XML cu datele introduse și il trimite la server pentru a se înregistra în sistem. Dacă totul a decurs bine, atunci se va recepționa un răspuns pozitiv la cererea de înrolare.

Panoul de verificare conține un ComboBox cu numele tuturor vorbitorilor înrolați, din care utilizatorul alege identitatea sa. Apoi utilizatorul va citi un singur enunț afișat într-o câmp de text, apăsând pe butonul ”Record” pentru a începe înregistrarea și pe acelși buton, acum cu denumirea ”Stop”, pentru a o opri. Apoi utilizatorul apasă pe butonul de ”Sign in” și așteaptă un răspuns de la server.

Panoul de verificare conține aceleași elemente ca și cel de verificare, dar lipsește ComboBox-ul. La apăsarea butonului de ”Sign in”, se va transmite doar fișierul audio înregistrat, nu și numele utilizatorului. Apoi se așteaptă răspunsul de la server.

Dacă verificarea sau identificarea s-a realizat cu succes, atunci se va afișa un alt panou cu câmpuri libere pentru nume, parolă, email și poză și câmpuri pline cu numărul enunțului și enunțul în sine. Utilizatorul are la dispoziție un buton de pornire/oprire a înregistrării și un buton de ”Sign out” pentru ieșire din sistem. Mai există și butonul de salvare a datelor care trimite datele din câmpurile completate și enunțurile înregistrate serverului sub forma unei cereri de actualizare a utilizatorului. Clientul afișază apoi răspunsul primit pentru această cerere.

4.4 Trainer

Este clasa care se ocupă de generarea modelelor pentru vorbitori. Ea conține patru metode, fiecare generând câte un fișier de care are nevoie următoarea:

- public static void initEM(String _databasePath,
  String _UBMFilePath,
  String _outputModelsFile,
  ArrayList<String> _trainingFileIDs)

Inițializează UBM-ul. Parametrii săi sunt calea către baza de date, calea către folderul ce va conține UBM-ul, numele fișierului UBM și o listă cu numele fișierelor ce vor fi folosite la antrenare.

- public static void trainEM(String _databasePath,
  String _UBMFilePath,
  String _inputModelsFile,
  String _outputModelsFile,
  ArrayList<String> _trainingFileIDs)

Definitivează UBM-ul. Parametrii săi sunt calea către baza de date, calea către folderul ce conține UBM-ul, numele fișierului UBM inițializat anterior, numele fișierului UBM final și lista fișierelor ce vor fi folosite la antrenare.

- public static void initMAP(String _databasePath,
  String _UBMFilePath,
  String _inputModelsFile,
  String _speakerGMMsPath,
  String _outputModelsFile,
  ArrayList<String> _trainingFileIDs)

Inițializează modelul pentru un vorbitor. Parametrii săi sunt calea către baza de date, calea către folderul ce conține UBM-ul, numele fișierului UBM final, numele folderului ce va conține modelul pentru vorbitor, numele fișierului cu modelul pentru vorbitor și lista fișierelor cu care se va face antrenarea.

- public static void trainMAP(String _databasePath,
  String _speakerGMMsPath,
  String _inputModelsFile,
  String _outputModelsFile,
  ArrayList<String> _trainingFileIDs)

Definitivează modelul pentru un vorbitor. Parametrii săi sunt calea către baza de date, numele folderului conține modele pentru vorbitori, numele fișierului cu modelul pentru vorbitor inițializat anterior, numele fișierului cu modelul final pentru vorbitor și lista fișierelor cu care se va face antrenarea.

4.5 Decoder

Conține o singură metodă, anume:

- public static void decode(String _databasePath,
  String _UBMFilePath,
  String _UBMFileName,
  String _speakerGMMsPath,

String _speakerGMMsName,
String _resultsPath,
ArrayList<String> _evaluationFileIDs,
LiumSpeakerRecognitionClientPeer _clientPeerReference)

Primește o listă cu numele unor fișiere și face identificarea lor folosind modelele indicate. Parametrii săi sunt calea către baza de date, calea către folderul ce conține UBM, numele fișierului UBM, calea către folderul ce conține modelele vorbitori, numele fișierului ce conține modelul sau modelele pentru fiecare vorbitor, folderul unde se vor scrie rezultatele, lista cu numele fișierelor ce vor fi identificate și o referință către client peer-ul apelant, pentru a transmite scorul maxim.

4.6 FileListBuilder

Conține două metode:

- public static ArrayList<String> buildUBMFileList(String _databasePath) – returnează o listă cu numele tuturor fișierelor audio din baza de date, cu excepția celor primite pentru identificare sau verificare. Calea către baza de date este dată ca parametru.
- public static ArrayList<String> buildTrainingFileList(String _databasePath, String _id) – returnează o listă cu numele tuturor fișierelor audio din baza de date care aparțin vorbitorului cu ID-ul indicat. Calea către baza de date este dată ca parametru.

4.7 XMLBuilder

Este compus din metode care încorporează datele primite în documentele XML prin intermediul cărora se realizează schimbul de informații dintre client și server. Toate metodele sunt publice, statice și returnează obiecte de tip Document.

- createGetEnrolledSpeakersRequest() – generează o cerere pentru lista vorbitorilor înrolați; nu are nevoie de date suplimentare
- createGetEnrolledSpeakersResponse(List<Speaker> _speakerList) – generează un document ce conține lista tuturor vorbitorilor înrolați, cu nume, parolă, email, poză și ID ca răspuns la o cerere tip GetEnrolledSpeakersRequest.
- createEnrollSpeakerRequest(Speaker _speaker, List<SpeechUtterance>, speechUtterancesList) – generează un document ce conține numele, parola, emailul, poza și ID-ul unui nou vorbitor împreună cu o listă de clipuri audio transmise sub formă de String prin utilizarea codării Base64
- createEnrollSpeakerResponse(boolean _response, String _responseDescription) – generează un document conținând răspunsul pozitiv sau negativ la o cerere de înrolare și o descriere mai detaliată a acestuia (de exemplu, utilizatorul nu a putut fi înrolat pentru că emailul ales este deja folosit)
- createSpeakerVerificationRequest(String _name, SpeechUtterance _speechUtterance) – generază un document ce conține numele vorbitorului și un enunț înregistrat și codat cu Base64
- createSpeakerVerificationResponse(boolean _response, String _responseDescription) – generează un document cu răspunsul pozitiv sau negativ la o cerere de verificare și o descriere a acestuia.

- createSpeakerIdentificationRequest(SpeechUtterance _speechUtterance) – generează un document ce conține un singur enunț înregistrat.
- createSpeakerIdentificationResponse(boolean _response, String _responseDescription) – generează un document ce conține răspunsul la o cerere de identificare împreună cu descrierea acestuia (de exemplu, numele vorbitorului identificat).
- createPasswordVerificationRequest(String _name, String _password) – generează o cerere de verificare a parolei compusă din numele utilizatorului și parola introdusă ce trebuie verificată.
- createPasswordVerificationResponse(boolean _response, String _responseDescription) – generează răspunsul la cererea de verificare a parolei.
- createDataUpdateRequest(Speaker _speaker, List<SpeechUtterance> _speechUtterancesList) – generează o cerere de actualizare a unui utilizator cu aceleași elemente ca la înrolare.
- createDataUpdateResponse(boolean _response, String _responseDescription) – generează răspunsul la o cerere de înrolare.
- createProtocolError(String _errorDescription) – genereză un mesaj de eroare.
- createSpeakerList(List<Speaker> _speakerList, int _nextId) – transformă o listă de vorbitori într-un obiect XML pentru a putea fi scris pe disc.

4.8 DataWriter

Metoda principală a acestei clase este:

- public static synchronized void write(List<SpeechUtterance> _speechUtteranceList,
  String _databasePath,
  String _speakerId,
  String _fileNamePattern)

Primește ca parametrii o listă de obiecte de tip SpeechUtterance, calea către baza de date, ID-ul vorbitorului și tiparul numelui fișierului. Fiecare fișier audio este decodat din forma Base64, este scris pe disc împreună cu transcrierea acestuia. Apoi este generat fișierul seg și în final sunt extrași parametrii MFCC folosind utilitarul Sphinx 4. Fișierele audio sunt mai întâi trecute print-un filtru trece-sus pentru a compensa pentru atenuarea de pe canal, apoi este împărțit în ferestre. Fiecărei ferestre îi este aplicată transformata Fourier rapidă, un banc de filtre Mel și transformata cosinus discreată. Rezultatele acestor operații sunt apoi scrise pe disc.

4.9 Speaker

Este clasa care reprezintă vorbitorii. Variabilele sale sunt:

- String name;
- String password;
- String email;
- String pictureFileName;
- String base64EncodedPicture;
- int id;

Toate variabilele sunt private, dar au metode pentru a le stabili și afla valorile. Apelarea funcției care returnează valoarea variabilei base64EncodedPicture face și codarea pozei în format

Base64, dacă aceasta nu este deja codată. Dacă variabila este nulă, atunci se codează fișierul a cărui cale este reținută de pictureFileName.

4.10 SpeechUtterance

Este clasa care reprezintă un enunț. Variabilele sale sunt:

- String text;
- String wavFileName;
- String base64Wav;

Variabilele sunt private, dar au metode pentru a le stabili și afla valoarea. La apelarea metodei care returnează valoarea variabilei base64Wav, dacă aceasta este nulă, se codează fișierul indicat de calea din wavFileName.

Concluzii

În acest proiect mi-am propus realizarea unei aplicații care să poată face verificarea sau identificarea automată a unui utilizator pe baza vocii sale. Am descris fundamentele recunoașterii vocale, diferențele dintre identificare și verificare și tipurile de recunoașteri. Am explicat ce înseamnă un model statistic al vorbitorului bazat pe mixturi Gaussiene pornind de la modelele Markov ascunse și am explicat ce sunt vorbitorii de fundal, împreună cu criteriile pentru alegerea acestora. În următorul capitol, am explicat principiile de bază a programării obiect-orientate și câteva aspecte specifice ale limbajului de programare Java, folosit în aplicația curentă.

Am descris utilitarul LIUM_SpkRecognition și la ce poate fi folosit el. Apoi am prezentat procesul prin care am extras funcțiile de care aveam nevoie și le-am integrat în proiectul creat de mine, punctând și problemele majore întâlnite pe parcurs și modul în care am reușit să le depășesc.

Apoi, în următorul capitol, am descris aplicația pe care am creat-o. Aceasta oferă unui utilizator mai multe opțiuni, prin intermediul clientului. Utilizatorul are la dispoziție o interfață grafică creată cu ajutorul librăriei Swing și se poate înrola, adică introduce numele, emailul și parola și înregistrează o serie de 10 enunțuri formate din 12 cifre generate aleator. Toate datele sunt transmise serverului, care adaugă utilizatorul la lista vorbitorilor deja existenți și generează un model pentru acesta, model care se bazează pe mixturi Gaussiene. Un utilizator deja înrolat se poate autentifica fie prin verificare, unde alege identitatea sa din lista vorbitorilor existenți, sau prin verificare, unde nu trebuie să își declare identitatea. În ambele cazuri, este nevoie de înregistrarea unui enunț. Acesta este transmis serverului, care compară trăsăturile extrase din enunț cu modelele stocate și ia decizia de acceptare sau respingere. În spatele acestei interfețe stau numeroase clase, atât pentru comunicarea dintre server și client, dar și pentru extragerea trăsăturilor, scrierea lor corectă pe disc, generarea modelelor, atât cel universal ce include toți vorbitorii, cât și cele particulare, proprii fiecărui utilizator. Am mai creat și clase necesare codării informației în documente XML.

Proiectul mi-a permis acumularea a numeroase cunoștințe, în special din domeniile recunoașterii vocale și al programării, și aplicarea lor pentru realizarea unei aplicații utile. Aceasta ar putea fi cu ușurință adaptată pentru a reglementa accesul într-o clădire sau accesul la un serviciu on-line. Se mai pot aduce îmbunătățiri prin incrementarea numărului de enunțuri per vorbitor necesare înrolării, ceea ce ar duce la modele mai exact pentru fiecare vorbitor, sau se poate realiza un client în alt limbaj decât Java deoarece schimbul de informații se face prin obiecte XML care nu sunt proprii Java.

Anexa 1

Tabelul 1. Caracteristicile bazelor de date cu vorbitori						
Baza de date	**Număr de vorbitori**	**Număr de enunțuri per vorbitor**	**Canal**	**Mediu acustic**	**Mediu de înregistrare**	**Interval între sesiuni**
TIMIT	630	10 fraze citite	Curat	Cabinade sunet	Microfon de bandă largă	Deloc
NTIMIT	630	10 fraze citite	Rețea de telefonie	Cabinade sunet	Button de carbon fix	Deloc
Switchboard	500	1-15 conversații	Rețea de telefonie	Acasă și birou	Variabil	Zile-Săptămâni
YOHO	138	4 pentru antrenare, 10 pentru testare	Curat	Birou	Telefon, calitate ridicat[	Zile-Luni

Tabelul 2. Experimentul de identificare cu Switchboard		
	Numere de telefon fără potriviri	**Numere de telefon cu potriviri**
Număr de enunțuri de test	74	398
Număr de erori	35	43
Procentaj de eroare	47.3%	10.8%

Tabelul 3. Experimente cu pretendenți și impostori pentru TIMIT și NTIMIT					
Experimente	**Număr de vorbitori**	**Număr de eunțuri proprii per vorbitor**	**Număr de enunțuri ale impostorilor per vorbitor**	**Număr total de eunțuri proprii per vorbitor**	**Număr total de enunțuri ale impostorilor per vorbitor**
M	112	2	202	224	22624
F	56	2	88	110	4945
M+F	168	2	313	334	52538

Tabelul 4. Experimente cu pretendenți și impostori pentru Switchboard					
Experiment	**Număr de vorbitori**	**Număr mediu de enunțuri proprii per vorbitor**	**Număr de teste cu enunțuri ale impostorilor per vorbitor**	**Număr total de teste cu enunțuri proprii**	**Număr total de teste cu enunțuri ale impostorilor per vorbitor**
M	12	4	210	47	2520
F	12	4	218	50	2612
M + F	24	4	428	97	10272

Bibliografie

[1] http://www.biometric-solutions.com/solutions/index.php?story=speaker_recognition
[2] http://www.samdrazin.com/classes/mmi361/project3.php
[3] http://www.scholarpedia.org/article/Speaker_recognition
[4] http://en.wikipedia.org/wiki/Speaker_recognition
[5] http://practicalcryptography.com/miscellaneous/machine-learning/guide-mel-frequency-cepstral-coefficients-mfccs/
[6] http://cs.brown.edu/research/ai/dynamics/tutorial/Documents/HiddenMarkovModels.html
[7] http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/RAJA/CV.html
[8] "A Review on Text-Independent Speaker Identification Using Gaussian Supervector SVM", Kauleshwar Prasad, Piyush Lotia, M. R. Khan
[9] "Fundamentals of Speaker Recognition", Homayoon Beigi, Springer, 2011
[10] J. Naik, "Speaker verification: A tutorial, IEEE Commun. Mag., vol. 28, pp. 42–48, Ian. 1990.
[11] "A new method of text-independent speaker recognition," A. L. Higgins, R. E. Wohlford, Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing, Tokyo, Japan, 1986, pp. 869–872
[12] Steve Renals, „Speaker Adaptation", Automatic Speech Recognition— ASR Lecture 11, March 2008
[13] "Phone Recognition on the TIMIT Database", Carla Lopes, Fernando Perdigão
[14] "NTIMIT: A Phonetically Balanced, Conrinuous Speech, Telephone Bandwidth Speech Database", C. Jankowski, A Kalyanswamy, S. Basson, and J. Spitz, Proc. Int. Conf. on Acoustics, Speech, and Signal Processing 1, Albuquerque, 3-6, Apr. 1990, p. 109.
[15] J. P. Campbell, Jr., "Testing with the YOHO CD-ROM voice verification corpus," Proc. IEEE Int. Conf. Acoustics, Speech, and Signal Processing, Detroit, MI, 1995, pp. 341–344
[16] http://en.wikipedia.org/wiki/Speaker_diarisation
[17] http://www.biometrics.gov/Documents/speakerrec.pdf
[18] http://www.informatik.uni-ulm.de/ni/Lehre/SS06/PraktikumNI/Campbell.pdf
[19] http://www.ms.sapientia.ro/~manyi/research/teza.pdf
[20] ahttp://www.busim.ee.boun.edu.tr/speechweb/index.php/research/9-projects/31-speaker-recognition
[21] "Automatic recognition of speakers from their voices, "Proc. IEEE, vol. 64, pp. 460–475, 1976
[22] "Automatic Speech and Speaker Recognition: Advanced Topics", Chin-Hui Lee, Frank K. Soong, Kuldip K. Paliwal, Kluwer Academic Publishers, March 31, 1996
[23] http://commons.wikimedia.org/wiki/Category:Hidden_Markov_Model
[24] http://www-lium.univ-lemans.fr/diarization/lib/exe/fetch.php/toolkit-interspeech2013.pdf
[25] http://www-lium.univ-lemans.fr/diarization/lib/exe/fetch.php/diarization-cmu-spud-2010.pdf
[26] Programming Language Popularity", http://www.langpop.com/
[27] "TIOBE Programming Community Index", http://www.tiobe.com, 2009
[28] "The Java Language Environment", James Gosling, Henry McGilton, Mai 1996.