Universitatea ‘’Politehnica’’ din București

Facultatea de Electronică, Telecomunicații și Tehnologia Informației

Serviciu web de transcriere automată a știrilor difuzate la radio

- Radio broadcast news transcription web-service-

Proiect de diplomă

prezentat ca cerință parțială pentru obținerea titlului de

Inginer in domeniul

Calculatoare şi Tehnologia Informaţiei

programul de studii de licență *Ingineria Informaţiei (CTI – INF)*

Conducător științific: Absolvent:

Lect. Horia CUCU Alexandru Constantin ȘERBAN

2014

DECLARAŢIE DE ORIGINALITATE A PROIECTULUI DE DIPLOMĂ/LUCRĂRII DE DISERTAŢIE[1)]

(MODEL)

Subsemnatul SERBAN V. ALEXANDRU CONSTANTIN [2)], posesor al B.I./C.I./paşaport seria KT, nr. 967385, având CNP 1 9 2 0 5 1 0 3 6 0 6 8 0, am întocmit proiectul de absolvire/diplomă/lucrarea de disertaţie cu tema:[3)] SERVICIU WEB DE TRANSCRIERE AUTOMATĂ A ȘTIRILOR DIFUZATE LA RADIO, în vederea susţinerii examenului de finalizare a studiilor universitare de licenţă/masterat, organizat de către Facultatea de ELECTRONICĂ, TELECOMUNICAŢII ŞI TEHNOLOGIA INFORMAŢIEI, Departamentul INFICFI, din cadrul Universităţii POLITEHNICA din Bucureşti, sesiunea SEPTEMBRIE, anul universitar 2014.

Luând în considerare conţinutul art. 143 din Legea Educaţiei Naţionale nr. 1/2011, al. (4) „*Îndrumătorii proiectelor de diplomă/lucrărilor de disertaţie răspund în solidar cu autorii acestora de asigurarea originalităţii conţinutului acestora*" şi al. (5) „*Este interzisă comercializarea de lucrări ştiinţifice în vederea facilitării falsificării de către cumpărător a calităţii de autor al unui/unei proiect de diplomă/lucrări de disertaţie*", declar pe proprie răspundere, că acest/această proiect/lucrare este original(ă), fiind rezultatul propriei activităţi intelectuale, nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei în vigoare.

Cunosc faptul că plagiatul sau prezentarea unui/unei proiect/lucrări, elaborat(ă) de alt absolvent sau preluată de pe internet, din manuale şi cărţi, fără precizarea sursei constituie infracţiune (furt intelectual şi nerespectarea dreptului de autor şi a proprietăţii intelectuale) şi atrage după sine anularea examenului de diplomă/disertaţie, precum şi răspunderea penală.

Data: 09.09.2014 Semnătura: (în original)

[1)] Declaraţia se completează „*de mână*" şi reprezintă un document care se depune la înscrierea pentru susţinerea examenului de finalizare a studiilor.

[2)] Numele, iniţiala prenumelui tatălui şi prenumele, cu majuscule.

[3)] Denumirea proiectului de diplomă/lucrării de disertaţie, cu majuscule.

Cuprins:

Lista figurilor și a tabelelor:

Lista de acronime :

BYTECODE
CROSS-PLATFORM
EXPRESSJS
DUMMY
EYES-FREE
FOSS
FRAMEWORK
FRONT-END
GNU GENERAL PUBLIC LICENCE (GPL)
GLASSFISH
HANDS-FREE
HEADER
HTTP - Hypertext Transfer Protocol
ISO / IEC JTC1
IETF - Internet Engineering Task Force
JAVA EE – Java Enterprise Edition
JAVA ME - Java Micro Edition
JAVA SE – Java Standard Edition
JRE - Java Runtime Environment
JSON - JavaScript Object NotationJ2SE
J2EE - Java 2 Platform Enterprise Edition
J2SE - Java 2 Platform Standard Edition
LOW-LEVEL
NoSQL - Not Only SQL
PassportJS
PULL MODEL
RESTful
SCORRER
SDK - Software development kit
SPI extensions - Serial Peripheral Interface extensions
SER - Sentence error rate
SSE – Server sent events
SOCKET
WEB
WEB-AS-CORPUS
WORD ERROR RATE
WORA - Write once, run anywhere

Introducere

Premisele proiectului și obiectivele principale.

Premisele proiectului au fost dezvoltarea unui sistem de recunoaștere automată a vorbirii continue, clare și coerente în limba română cu ajutorul unui limbaj de programare de nivel înalt și prezentarea rezultatului printr-un protocol de comunicare ce respectă standardele web actuale. Obiectivul final este testarea acestui sistem prin achiziția de date provenite de la diverse posturi de radio din România și prezentarea transcrierii către un solicitant web. Deoarece posibilitățile in acest domeniul sunt foarte vaste, este de dorit ca prezentarea datelor să poată satisface cerințele de scalabilitate pe care le denotă protocolul de comunicare. De asemenea, având în vedere varietatea actuală a posturilor radio, arhitectura sistemului de recunoaștere a fost concepută pentru a putea îndeplini orice solicitare. Este de menționat faptul că scopul este cel de recunoaștere a vorbirii continue, astfel încât domeniul de activitate al unui post de radio ar trebui să fie orientat către acest domeniu(știri, conferințe, interviuri) și nu către muzică .

Gradul de noutate al proiectului este adus de alegerea limbii române pentru efectuarea transcrierii. Deoarece necesitatea unei baze de date cu înregistrări în această limbă este crucială, foarte puține proiecte de recunoaștere sunt implementate și dezvoltate actual în România.

Cele mai multe resurse în acest domeniu au fost create de către grupuri de cercetare și nu sunt ușor accesibile.

Realizarea proiectului a presupus adaptarea unui sistem de recunoaștere CMU Sphinx pentru îndeplinirea cerințelor de achiziție de semnal audio, procesare a acestuia și prezentarea rezultatelor ca un serviciu web de sine stătător.

Pentru realizarea sistemului de recunoaștere a vorbirii, au fost puse la dispoziție resursele de tip model acustic și lingvistic necesare implementării; dezvoltarea sau îmbunătățirea acestora nu a fost obiectivul proiectului curent. Pentru o introducere în tehnica de recunoaștere a fost dezvoltat un proiect cu sarcina de recunoaștere a cifrelor limbii române. Acest proiect a presupus dezvoltarea modelelor acustice și lingvistice necesare și a unei gramatici aferente, scopul său fiind familiarizarea cu tehnica de recunoaștere a vorbirii, arhitectura generală a unui astfel de sistem de recunoaștere a vorbirii și tehnici de evaluare. Rezultatele acestui proiect sunt prezentate, de asemenea, în lucrarea curentă. Cu ajutorul resurselor puse la dispoziție, după familiarizarea cu sarcina de recunoaștere a vorbirii continue, s-a impus adaptarea sistemului de recunoaștere ce folosește resursele prezentate mai sus la achiziția de semnal audio prin intermediul stream-urilor radio și dezvoltarea unei resurse capabile să stocheze pe disc transmisia radio. Această resursă este folosită în scopul evaluării și corectării transcrierii.

Deoarece tema proiectului presupune implementarea sistemului de transcriere ca un serviciu web, aportul personal este bazat mai mult pe implementarea unei arhitecturi capabile să prezinte rezultatele transcrierii prin intermediul protocolului HTTP. Sistemul poate comunica, de asemena,

și prin intermediul Java sockets. Din păcate această arhitectură este capabilă să comunice prin intermediul obiectelor Java dezvoltate în partea de arhitectură și nu satisface standardele de scalabilitate impuse. Datorită adaptării sistemului la comunicația HTTP, orice client web se poate conecta și beneficia de transcrierea aferentă, pentru obținerea acesteia fiind necesar doar un browser web.

Un alt obiectiv important a fost dezvoltarea unei interfețe grafice capabile să permită utilizatorului să asculte postul de radio ales și să beneficieze în același timp de transcriere.

Forma finală a proiectului este un pachet web ce dispune de autentificare, prezentare și demo-ul aferent. Pentru implementare au fost folosite diferite tehnologii web prezentate în capitolele următoare.

Analizând sarcina de recunoaștere a vorbirii continue prin achiziția semnalului de la diferite posturi radio și având în vedere conținutul difuzat de către posturile radio, s-a impus dezvoltarea unei componente ce poate elimina momentele de liniște, zgomot sau muzică și înaintarea către sistemul de recunoaștere a semnalului audio ce conține doar discurs (vorbire liberă, dialog, etc.).

Capitolul 3 din acest document prezintă progresiv activitatea de dezvoltare desfășurată pentru îndeplinirea sarcinilor și a obiectivelor propuse.

Capitolul 1. Introducere in tehnica de recunoaștere a vorbirii directe. Prezentarea metodei și a tehnicii folosite in implementare.

1.1 Introducere în recunoașterea automată a vorbirii.

Recunoașterea automată a vorbirii presupune achiziția de semnal audio ce conține vorbire și transformarea acestuia într-o secvență text privită ca o succesiune de cuvinte. În cazul în care semnalul audio procesat conține secvențe de dialog provenite de la mai mulți vorbitori, putem privi procesul de recunoaștere automată a vorbirii ca având două etape: a) recunoașterea vorbitorului și atribuirea secvenței vorbite acestuia (diarizare) și b) transcrierea efectivă a dialogului ce presupune transformarea semnalului audio în text.

Domeniile de aplicabilitate ale recunoașterii de vorbire sunt auto-explicative. Putem privi cazul interfețelor hands-free sau eyes-free; caz în care utilizatorul nu poate folosi alt mijloc de comunicare decât vorbirea (trebuie să avem în vedere și faptul că vorbirea este cel mai natural mijloc de comunicare pentru ființele umane). De asemenea, recunoașterea automată a vorbirii poate fi folosită și pentru transcrierea unui monolog al unui vorbitor (dictare). Această transcriere este utilă în diferite domenii: editarea documentelor în cazul jurnaliștilor sau a scriitorilor, transcrierea proceselor juridice, transcrierea unor conferințe, cursuri, etc. Aceste domenii sunt, totuși, mult mai restrictive decât problema generală ce presupune transcrierea vorbirii naturale, continue, provenite de la un vorbitor necunoscut. Un factor foarte important este mediul de proveniență al semnalului (de multe ori semnalul este înregistrat într-un mediu zgomotos). Variabilitatea, în multe situații practice, este restricționată (vorbitorul ar putea fi cunoscut/necunoscut, vorbirea ar putea fi dictată/spontană, mediul de înregistrare ar putea fi zgomotos, reverberant sau total opus). Pentru recunoașterea automată a vorbirii, se face o distincție majoră între modul în care este abordată variabilitatea acustică și modul în care se interpreteaza incertitudinea lingvistică (modelul de limbă).

Un alt factor foarte important ce intervine în dificultatea procesului de transcriere este sarcina de recunoaștere. Aceasta este descrisă de specificitatea limbii, numărul de cuvinte ce pot fi pronunțate și incertitudinea lingvistică a sarcinii de recunoaștere. Diverse limbi vorbite reprezintă o provocare diferită pentru un sistem de recunoaștere. Pentru un număr foarte mare de limbi nu există resurse acustice (o bază de date de vorbire) sau lingvistice. Având în vedere dependența proiectării și dezvoltării unui sistem de recunoaștere automată a vorbirii de aceste resurse, rezultatul obținut este direct dependent de complexitatea și acuratețea cu care aceste resurse sunt colectate.

Complexitatea morfologică a limbilor vorbite este un alt factor decisiv în proiectarea și implementarea unui sistem de recunoaștere a vorbirii. Limba română prezintă o complexitate și diversitate ridicată: vocabularul are dimensiuni mari, semnificativ mai mari decât alte limbi de circulație internațională. De asemenea, gramatica limbii române este un alt factor cheie în proiectarea unui astfel de sistem. Limba română dispune de o varietate foarte mare de timpuri și

persoane ale verbelor, această varietate îngreunând semnificativ dezvoltarea unui sistem de recunoaștere automată a vorbirii.

Privind din punct de vedere al vocabularului, putem compara cazul unui sistem de recunoaștere a cifrelor (ce cuprinde un vocabular de doar 10 cuvinte) cu sarcina exercitată de un sistem de recunoaștere a vorbirii continue(ce cuprinde un vocabular de 64.000 de cuvinte). Cu toate acestea, acest factor nu este definitoriu pentru complexitatea unui sistem de recunoaștere. Incertitudinea lingvistică a discursurilor potențiale ce urmează să fie recunoscute este un factor mult mai important.

Un alt factor ce influențează dificultatea procesului de recunoaștere este modul (stilul) vorbirii. Acesta este descris de fluența cu care este vorbită limba ce urmează a fi recunoscută, naturaleţea sau strictețea cu care sunt respectate normele și standardele acesteia (în vorbirea conversațională se pot face diferite excepții).

Astfel, putem clasifica sarcinile de recunoaștere a vorbirii continue în funcție de dificultatea lor: sarcinile de recunoaștere a vorbirii cititie au o dificultate mică comparativ cu sarcina de recunoaștere a vorbirii conversaționale sau spontane unde variabilitatea acustică este factorul cel mai important. Acuratețea procesului de recunoaștere este principala caracteristică influențată de mediul acustic în care este înregistrată vorbirea și de canalul de transmisie. În afara mediilor de înregistrare dedicate (studiouri de înregistrare, laboratoare de cercetare, etc) semnalul rpovine , de obicei, de la mai multe surse acustice, diferiți vorbitori, etc. În majoritatea cazurilor separarea semnalelor acustice diferite reprezintă o problemă. Mediul de înregistrare (microfonul) are un impact semnificativ asupra factorului acuratețe.

Sistemele comerciale de dictare alături de proiectele de cercetare în acest domeniu sunt realizate cu microfoane de înaltă calitate, dar, de obicei, înregistrarea se face cu mijloace de calitate medie, chiar joasă. Variațiile în canalul de transmisie (în cazul nostru aerul) apar odată cu mișcările corpului, ale capului relativ la microfon și prin transmisia prin diferite medii(internet). Cea mai mare varietate apare în situația în care semnalului acustic i se adaugă zgomot aditiv mare, sau când acesta provine de la mai multe surse acustice. Recunoașterea semnalului cu un raport semnal-zgomot scăzut, poate avea rezultate de 2 până la 4 ori mai slabe comparativ cu recunoașterea semnalului curat.

Trebuie, de asemenea, luate în considerație și caracteristicile vorbitorilor. Limba sau dialectul folosit, vorbitorul nativ sau nu, rapiditatea pronunției, vârsta vorbitorului sau diferențele anatomice și fiziologice influențează producerea vorbirii. Vorbitori diferiți au, de asemenea, grade diferite de variabilitate intrinsecă, bazate pe starea emoțională, probleme de sănătate, etc. Această problemă poate fi rezolvată prin proiectarea de sisteme dependente de vorbitor. Evident, complexitatea unui astfel de sistem crește proporțional cu numărul de vorbitori implicați.

În cazul concret, al sistemului de recunoaștere al vorbirii provenite de la posturile de radio, varietatea vorbitorilor fiind uriașă, nu este util să identificăm și salvăm proprietăți despre vorbitor. De asemenea, în acest caz, mediul de transmisie este unul profesionist. Este de dorit ca un radio să dispună de un studio de difuziune ce dispune de echipament de înregistrare de înaltă calitate și un mediu acustic izolat corespunzător. Principalele dificultăți întâmpinate în proiectare provin din compresia semnalului acustic înainte de transmiterea pe diferite canale și de varietatea de sunete adăugate semnalului vocal (deseori există muzică pe fundal, diferite sunete ce întrepătrund

discursul, etc.). De asemenea, calitățile vorbitorilor diferă, este de anticipat ca un prezentator să aibă un discurs cursiv și rapid, dar există foarte multe momente în care vorbitorul liber este un invitat (cazul unui interviu), iar proprietățile ce descriu discursul acestuia diferă foarte mult. De asemenea, o proprietate a discursului din acest mediu o reprezintă rapiditatea cu care este rostit (exemplul unui crainic radio ce prezintă știrile este foarte relevant în acest caz).

Paradigma state-of-the-art pentru recunoașterea vorbirii continue cu vocabular extins o reprezintă modelul Markov ascuns(Hidden Markov Model - HMM). Această infrastructură a fost introdusă în anul 1975 de către Baker și reprezintă un candidat viabil pentru modelarea acustică a recunoașterii vorbirii. În particular, acest model este îmbinat cu modele de tip n-gram, responsabile pentru modelarea limbii. Modelele lingvistice statistice au devenit foarte utilizate odată cu extinderea internetului, fapt ce a furnizat date suficiente pentru antrenarea acestor sisteme.[1]

1.2 Modul de funcționare al unui sistem de recunoaștere automate a vorbirii

Acest proces presupune transformarea unui semnal audio într-o succesiune de cuvinte. În acest domeniu, tehnicile de modelare statistică (crearea modelelor pe baza unor cantități mari de date) s-au impus ca și standard. Platforma statistică pentru recunoașterea automată a fost creată și dezvoltată de echipe de cercetare din laboratoare ale companiilor IBM sau AT&T.

Acest proces poate fi formulat statistic ca fiind cea mai probabilă secvență de cuvinte W* într-o anumită limbă, dat fiind mesajul X.

Reprezentarea formală utilizează funcţia argmax, care selectează argumentul ce maximizează probabilitatea secvenţei de cuvinte:

$$W^* = \arg\max_W p(W \mid X) = \arg\max_W \frac{p(X \mid W)p(W)}{p(X)} = \arg\max_W p(X \mid W)p(W)$$

Fig 1. 1 Reprezentarea unei secvențe de cuvinte dintr-o limbă dat fiind mesajul X

Dezvoltarea din această ecuație are la bază regula Bayes şi este facută ținând cont de faptul că probabilitatea mesajului vorbit p(x) este independentă de secvența de cuvinte W. Ultimul rezultat evidenţiază doi factori care pot fi estimaţi direct. Problema iniţială (găsirea secvenţei de cuvinte pe baza mesajului vorbit) a fost împărţită în două sub-probleme mai simple: a) estimarea probabilităţii apriori a secvenţei de cuvinte p(W) şi b) estimarea probabilităţii mesajului vorbit dată fiind secvența de cuvinte pronunţată p(X|W). Primul factor poate fi estimat utilizând exclusiv un model de limbă, iar cel de-al doilea poate fi estimat cu ajutorul unui model acustic. Cele două modele pot fi construite independent aşa cum se va vedea în secţiunea următoare, dar vor fi folosite împreună pentru a decoda un mesaj vorbit, aşa cum arată Ecuaţia 1.1. [2]

[1] Research and Development Project in Spoken Language Technology – Horia Cucu
[2] http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-345-automatic-speech-recognition-spring-2003/lecture-notes/

1.3 Arhitectura generală a unui sistem de recunoaștere automată a vorbirii

Următoarea figură prezintă arhitectura generală a unui sistem de recunoaștere automată a vorbirii:

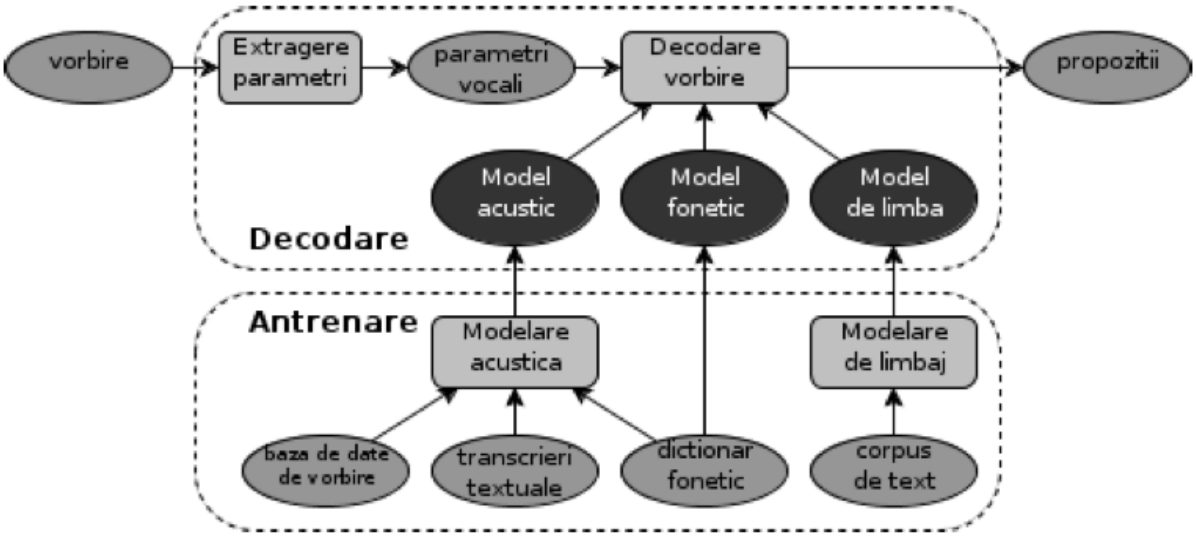

Fig 1. 2 Arhitectura generală a unui sistem de recunoaștere a vorbirii

Arhitectura generală se bazează pe recunoașterea vorbirii folosind o serie de parametri vocali ce se extrag din semnalul vocal și pe baza unor modele dezvoltate în prealabil (model acustic, fonetic, de limbă).

Modelul acustic estimează probabilitatea mesajului vorbit într-o succesiune de cuvinte. Unitatea acustică de bază folosită în acest caz nu este cuvântul deoarece numărul acestora într-o limbă este foarte mare. Se folosesc unități acustice sub-lexicale (exemple fiind foneme sau senonele). Aceste unități sub-lexicale se conectează în timpul procesului de decodare și formează modele pentru cuvinte și pentru succesiuni de cuvinte. Modelele acustice estimează probabilitatea p(X|W) descrisă mai sus și implementată cu ajutorul modelelor Markov.

Modelele Markov sunt definite ca automate probabiliste cu un număr finit de stări ce pot fi combinate într-un mod ierarhic pentru a construi modele acustice coespunzătoare unor secvențe de cuvinte. Ele nu modelează un semnal acustic folosind forma de undă a acestuia. Un bloc de extragere de parametri este folosit pentru a calcula anumiți coeficienți ce sunt modelați de către modelul acustic. Semnalul audio inițial în domeniul timp este transformat într-o secvență de ferestre în același domeniu. Din aceste ferestre se extrag parametri de tip spectral sau cepstral. Cel mai folosit model este cel de tip cepstral-perceptual (exemplu Mel-cepstral) și parametri obținuți prin predicție liniară (PLP - Perceptual Linear Prediction). Avantajul particular al reprezentării cepstrale este decorelația coeficienților . Extragerea acestor parametri este ilustrată în următoarea figură :

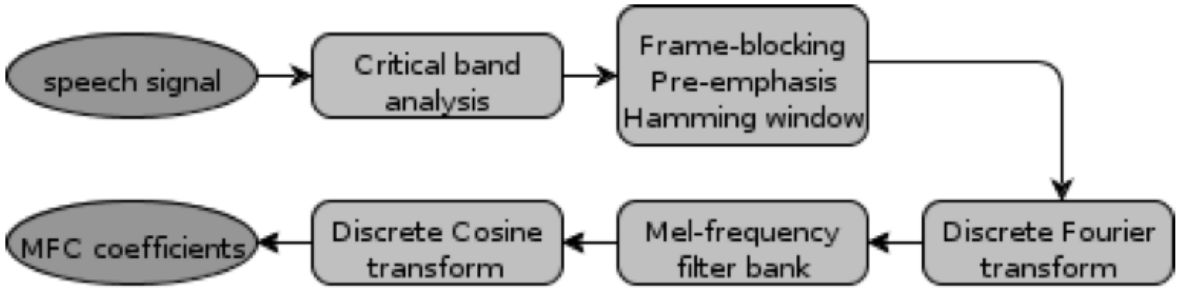

Fig 1. 3 Extragerea parametrilor MFCC

Deși fiecare set de coeficienți este calculat doar pe o scurtă fereastră, informația este utilă în sistemele de recunoaștere a vorbirii. Energia acestor parametri și derivatele temporare sunt de asemenea utile. Cel mai utilizat set de parametri este format din 39 de coeficienți; 12 MFCC + energie, la care se adaugă derivate temporale de ordin unu și doi.

Așa cum am spus, un model Markov ascuns este un automat cu stări finite ce conține un set de stări conectate reprezintând diferite tranziții. Secvențele de stări sunt ascunse, astfel încât procesul este considerat ascuns : această secvență de stări nu este disponibilă unui observator.

Modelul Markov ascuns este ilustrat în următoarea figură :

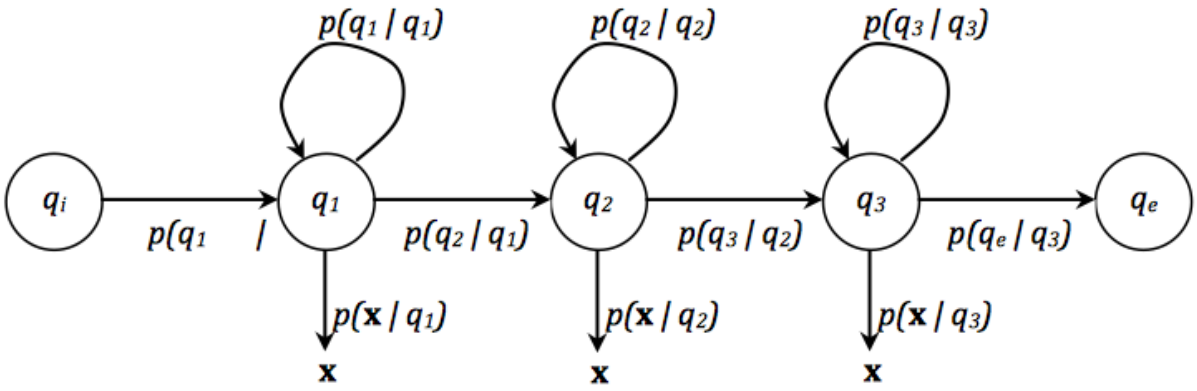

Fig 1. 4 Reprezentarea unui model Markov ascuns drept automat cu număr finit de stări

Putem observa că un model Markov poate fi caracterizat de parametri:

- setul de stări Q=$q1q2\ldots qN$;

- setul de probabilități A=a11a12..aNN ce reprezintă probabilitățile de tranziție între stări

- setul de observații probabilistice B=bi(xt)=p(xt|qi), ce exprimă probabilitatea ca observația xt să fie generată în starea i.

Modelarea vorbirii cu ajutorul acestor modele presupune ca tranziția între stările unui proces ascuns să fie un proces Markov de prim ordin, în care probabilitatea tranziției depinde doar de

starea curentă și probabilitatea ca o stare să genereze un anumit vector să fie independentă de vectorii de parametri generați .[34]

Modelul fonetic intervine în conectarea modelului acustic (estimarea probabilităților acustice ale fonemelor) cu modelul de limbă (estimarea probabilităților secvențelor de cuvinte). Deseori acest model este un dicționar de pronunție ce asociază fiecărui cuvânt din vocabular una sau mai multe secvențe de foneme adecvate reprezintând modul de pronunție a respectivului cuvânt. Așadar, dicționarul fonetic stă la baza dezvoltării modelului fonetic necesar în procesul de decodare. În cazul în care acest dicționar cuprinde mai multe pronunții pentru un același cuvânt, pentru modelul fonetic aceste pronunții formează împreună un model de pronunție pentru respectivul cuvânt.

Un dicționar fonetic reprezintă un instrument linvistic ce specifică modul de pronunție al cuvintelor unei limbi. Acesta face corespondența între forma scrisă și cea fonetică a cuvintelor. Forma fonetică a cuvintelor reprezintă o succesiune de foneme. Într-un sistem de recunoaștere, rolul dicționarului este de a face legătura între modelul acustic și modelul de limbă. Așadar, acesta trebuie să conțină toate cuvintele posibile pentru sarcina de recunoaștere și transcrierea asociată acestora.

Procesul de construcție al dicționarului presupune crearea formei fonetice a cuvintelor unui vocabular necesar sarcinii de recunoaștere a vorbirii. Regulile acestui proces pentru limba română au fost studiate foarte puțin. Orice proces de construcție al unui dicționar fonetic va trebui să înceapă cu stabilirea acestor reguli de fonetizare și a excepțiilor aferente.

Construcția unui dicționar poate fi realizată manual, semi-automat sau chiar automat. Dacă vocabularul presupus unei sarcini de recunoaștere este foarte mic, acesta poate fi realizat manual, dar pentru un dicționar extins presupus unei sarcini de recunoaștere foarte mari, fonetizarea manuală nu reprezintă o opțiune.

În limba română, corespondența dintre forma scrisă și cea pronunțată este bijectivă. Există de asemenea și clase de ambiguitate în care aceeași literă are o pronunție diferită dependentă de context. Aceste clase de ambiguitate sunt studiate separat în momentul alcătuirii dicționarului.

Odată cu realizarea setului de reguli, fonetizarea se poate realiza automat prin folosirea unor aplicații software ce implementează setul de reguli de fonetizare stabilit a priori. Indiferent dacă, la o primă privire, acest lucru nu prezintă o dificultate mare, implementarea acestor reguli este o sarcină dificilă, deloc trivială. Pentru rezolvarea claselor de ambiguități descrise mai sus, este nevoie de informații suplimentare, cum ar fi modul de despărțire al cuvintelor în silabe. În cazul procesului de fonetizare manual, factorul uman ce cunoaște aceste reguli de despărțire joacă un rol crucial, însă, in cazul procesului automat, formalizarea acestor reguli este absolut necesară.

O alternativă viabilă pentru procesul de fonetizare automat sunt aplicațiile de învățare automată bazate pe model statistic. Utilizând dicționare fonetice create manual, putem antrena un model statistic de fonetizare, ce poate fi utilizat pentru fonetizarea unor vocabulare mult mai extinse. Este evident că acest proces creează dependențe de noi factori de studiu și cercetare privind diferite metode de abordare.

[3] http://www.voxforge.org/home/docs/faq/faq/what-is-an-acoustic-model
[4] http://cmusphinx.sourceforge.net/wiki/tutorialam

Unitatea de sunet fundamentală din limbile vorbite ce ajută la diferențierea cuvintelor și morfemelor este fonemul. Prin modificarea acestei părți a unui cuvânt se poate genera, după caz, fie un cuvănt ce nu există, fie cuvânt cu alt sens. [5]

În limba română se folosesc 7 vocale de bază, 2 imprumutate, 4 semi-vocale și 22 de consoane.

Pentru antrenarea unui model acustic a unui sistem de recunoaștere, se folosește o bază de date de vorbire completă. Acest model de bază de date va trebui să conțină trei componente dependente. Se impune existența unui set de clipuri ce conțin vorbire, a unui set de fișiere text ce vor conține textul pronunțat în aceste clipuri audio și, după caz, marcaje temporale pentru fiecare cuvânt sau fonem și eventuale informații suplimentare privint stilul și domeniul vorbirii. La aceste informații pot fi adăugate informații specifice vorbitorului curent, etc.

În general, clipurile audio ce conțin vorbirea vor fi eșantionate la o frecvență de 16kHz; 16 biți pentru dimensiunea eșantioanelor, formatul fișierelor fiind mswav.

Crearea bazei de date folosită pentru antrenarea unui sistem este un element foarte important ce afectează în mod direct performațele unui sistem de recunoaștere, parametrii de evaluare fiind în acest caz rata de eroare și viteza de recunoaștere. Caracteristicile cele mai importante ale unei baze de date de vorbire sunt, în primul rând, dimensiunea acestei baze de date exprimată prin numărul de ore de vorbire continuă sau numărul de vorbitori. O altă caracteristică importantă este stilul vorbirii. Așa cum a fost descris de-a lungul acestui document, stilul vorbirii poate fi reprezentat prin cuvinte izolate, vorbire continuă citită (dictată) sau vorbire liberă, conversațională. Factorul de variantă sau variabilitate este de asemenea foarte important în descrierea bazei de date. Calitatea înregistrărilor, zgomotul aferent acestora sau variabilitatea vorbitorilor pot influența în mod direct calitatea bazei de date. Trebuie avut în vedere foarte clar faptul că, această bază de date este folosită pentru antrenarea sistemului, astfel încât, pentru performațe bune, acești factori trebuie tratați cu cea mai mare atenție.

Modelarea acustică este abordată statistic, așadar, dimensiunea bazei de date reprezintă un factor cheie pentru sarcina de recunoaștere. Sarcina de comandă și control reprezintă o sarcină tipică în recunoașterea vorbirii pseudo-continue ce necesită un vocabular redus, iar sarcina de dictare este tipică recunoașterii vorbirii continue necesitând un vocabular extins.

Modelul propus, independent de vorbitor este un model foarte dificil de atins, dimensiunea bazei de date de antrenare crescând considerabil în acest caz. Este estimat că materialul vorbit trebuie să provină de la cel puțin 200 de vorbitori diferit. Variabilitatea poate fi atinsă doar prin înregistrarea mai multor vorbitori.

Crearea acestei baze de date presupune, în primul rând, înregistrarea unor texte predifinite. Pe lângă alegerea materialului ce urmează a fi înregistrat și a diferiților parametri ce țin de procesul de înregistrarea (locul, mijlocul de înregistrare, etc.), la crearea bazei de date se vor avea în vedere o

[5] [Bahl, 1991] Bahl, L.R., et al., "Multonic Markov Word Models for Large Vocabulary Continuous Speech Recognition," IEEE Transactions on Speech and Audio Processing, 1993, vol. 1, no. 3, pp. 334-344, 1991.

sumă de alți factori foarte importanți. În primul rând frazele vor trebuie să prezinte un material relevant din punct de vedere lexical și gramatical. Vor trebui evitate schimbările de topică a propozițiilor. Dimensiunea frazelor va trebui să fie scurtă pentru a putea fi pronunțate fără ezitări sau respirații. Conținutul frazelor va fi reprezentat de cuvinte uzuale în limbajul de zi cu zi, pentru a putea fi pronunțate ușor. Conectarea semantică a frazelor ajută vorbitorul să înțeleagă foarte repede sensul acestora, astfel putându-se obține o fluență în pronunție. Se presupune că frazele vor conține doar cuvinte specifice limbii dorite . Respectarea semnelor de punctuație, evitarea numerelor scrise cu cifre, a interjecțiilor este de dorit în cadrul înregistrărilor.

Avantajele acestei metode de înregistrare sunt date de controlul asupra materialelor înregistrate, a calității și a variabilității. Achiziția se poate face într-un interval de timp relativ scurt, iar cei implicați nu vor trebui antrenați în acest sens. Erorile de pronunție vor fi, de obicei, rare, astfel încât baza de date se poate apropria de corectidudine.

Un dezavantaj clar este dat de fluctuațiile foarte puține ale înregistrărilor astfel ca nu se pot obține baze de date pentru vorbire spontană sau conversațională.

Etichetarea materialelor obținute în urma vorbirii presupune fragmetarea materialelor audio înregistrate la aceeași frecvență de eșantionare, selectarea părților ce conțin vorbire și etichetarea acestora.

Posibilitatea obținerii unei baze de date de vorbire spontană este iminentă în acest caz deoarece înregistrarea presupune diferiți vorbitori, diverse contexte și diverse stări emoționale sub care se află vorbitorii. Prețul obținerii unei astfel de baze de date este totuși foarte ridicat, fiind influențat de dificultatea procesului de achiziție, procesare și etichetare. Timpul necesar etichetării unei ore de vorbire spontană este de cel puțin 3-4 ori mai mare decât timpul necesar înregistrării. Etichetarea nu poate fi făcută fără un antrenament, erorile de etichetare având o frecvență mai mare decât erorile de pronunție.[6]

În sistemele de recunoaștere automată a vorbirii directe de comandă sau control, extragerea unui corpus de text pentru antrenare nu este necesară deoarece aceste sisteme se folosesc de gramatici cu reguli. În cazul recunoașterii vorbirii continue însă, necesitatea unui vocabular extins bazat pe modele de limbă statistice, introduce necesitatea existenței corpusurilor de text de dimensiuni cât mai mari și mai adaptate domeniului din care fac parte mesajele ce urmează a fi decodate.

Achiziția unui corpus de text de dimensiuni mari (milioane-miliarde de cuvinte) nu poate fi făcută decât în mod automat, cantitatea de text în acest caz putând fi gasită pe internet. Așadar, metodele de dezvoltare a corpusurilor de text folosesc principiul numit Web-as-Corpus ce implică identificarea, descărcarea și procesarea materialelor text.

Normalizarea și restaurarea corpusului de text presupune procesarea acestuia pentru a obține un corpus de text util antrenării modelului lingvistic al sistemului de recunoaștere de vorbire

[6] [Cucu, 2011] Cucu, H., "Towards a speaker-independent, large-vocabulary continuous speech recognition system for Romanian," Teză de doctorat, Universitatea Politehnica din Bucureşti, România, 2011

continuă. Un astfel de sistem are rolul de a transcrie vorbirea, textul final va conține numai cuvinte și simboluri cu corespondent acustic. Modelul de limbă va trebui să modeleze numai probabilitățile de apariție a cuvintelor și a succesiunilor de cuvinte pronunțabile.

Având în vedere sursa de achiziționare a textului, normalizarea și restaurarea poate presupune diverse operații, dintre care putem menționa o serie de operații indispensabile : eliminarea etichetelor HTML în cazul achiziționării de pe internet, normalizarea caracterelor cu caracter diacritic, expandarea abrevierilor, transcrierea numerelor scrise cu cifre în numere scrise cu litere, eliminarea sau înlocuirea semnelor de punctuație si a caracterelor speciale (virgulele, punctele, semnele de întrebare vor trebuie înlocuite cu caracterul de linie nouă, parantezele vor trebui șterse, liniuțele, cu excepția celor din interiorul cuvintelor vor trebui șterse, iar caractere de tip procent vor trebui înlocuite cu expresia la sută). Se impune, de asemenea, înlocuirea tuturor literelor mari cu litere mici.

Așadar, sistemele de recunoaștere a vorbirii continue actuale utilizează modele statistice pentru modelarea probabilităților de apariție a cuvintelor și a succesiunilor de cuvinte din cadrul limbii dorite. Antrenarea modelelor statistice presupune cantități mari de date reprezentative pentru fenomenul ce urmează a fi modelat. Sistemele de recunoaștere a vorbirii continue presupun baze de date de vorbire etichetată în vederea modelării pronunției fonemelor sau a corpusurilor de text în modelarea statisticii apariției cuvintelor.[7]

Modelul de limbă este ultimul parametru absolut necesar prezentat în figura 1.3. Rolul acestuia este de a estima probabilitatea unei secvențe de cuvinte W = w1, w2, …, wn să fie o secvență validă pentru sarcina de recunoaștere. Rezultă că problema estimării unei probabilități a secvenței de cuvinte W este împărțită în mai multe probleme de estimare a probabilității unui cuvânt relativ la secvența de cuvinte anterioara. Isoriile de cuvinte precedente nu includ un număr indefinit de cuvinte, deoarece sarcina computațională ar crește proporțional. Acest număr este ales drept un factor m. Se presupune că doar un număr limitat de cuvinte induce probabilitatea unui cuvânt următor. Acest lucru conduce la modelul de limbă convențional definit și n-gram. În mod normal, factorul descris m este ales relativ la cantitatea de text disponibilă procesului de antrenare. De obicei se folosesc modele de limbă de tip trigram, acestea luând în considerație doar ultimele două cuvinte în prezicerea celui de-al treilea. Este nevoie de colectarea unor statistici de apariție a secvențelor de trei cuvinte (trigrame). Există posibilitatea construirii modelelor de limbă cu doar două cuvinte (bigrame) sau a secvențelor cu format mai mare (ordin superior).[8]

Modelul de limbă de tip n-gram se construiește prin estimarea probabilităților exprimate mai sus cu ajutorul unor corpusuri de text cu dimensiuni mari. În cazul unui model bigram, vor trebui estimate probabilitățile p(wj|wi) succesiv fiecărei perechi de cuvinte (wi,wj). Calcularea acestor probabilități este realizată cu ajutorul principiului maximul likelihood; se va număra de câte ori cuvântul wi este urmat de cuvântul wj comparativ cu alte cuvinte. Estimarea acestor probabilități necesită o cantitate mare de date de antrenare (de ordinul a câteva zeci de milioane de cuvinte).

Pentru construcția modelelor de ordin superior această cantitate va crește proporțional.

---

[7] [Cucu, 2011] Cucu, H., "Towards a speaker-independent, large-vocabulary continuous speech recognition system for Romanian," Teză de doctorat, Universitatea Politehnica din Bucureşti, România, 2011

[8] http://en.wikipedia.org/wiki/Language_model

Dezavantajul întâlnit la construcția modelelor de tip n-gram este dat de faptul că, indiferent de dimensiunea corpusului de antrenare vor exista n-grame ce nu vor apărea în acest text, însă ce pot apărea în textul de evaluare. Probabilitățile ce au fost estimate pe baza numărului de apariție a n-gramelor în corpusculi vor trebui ajustate. Metodele ce țin de acest proces de ajustatare poartă numele de metode de netezire și se bazează pe extragerea unei părți din probabilitatea ce se alocă n-gramelor întâlnite la antrenare și redistribuirea acesteia n-gramelor necunoscute.[9]

Din punct de vedere al performanței unui model de limbă, cel mai important criteriu este rata de eroare la nivelul unui cuvânt (word error rate). Dacă nu este implicat un sistem de recunoaștere automată a vorbirii, puterea de predicție a unui model de limbă poate fi evaluată măsurând probabilitatea acestuia asignată unor secvențe de cuvinte de evaluare. Un model de limbă este considerat viabil dacă atribuie o probabilitate mare unui text corect și o probabilitate mică unui text considerat deficitar. Acest caz este atribuit unui alt criteriu de măsurare a performanței definit ca perplexitate. Perplexitate mare pe o secvență de cuvinte denotă o capacitate de predicție mică pentru un model de limbă.

Există posibilitatea ca unele cuvinte conținute de textul ce urmează a fi evaluat să nu fie întâlnite în corpusul de antrenare. În acest caz aceste cuvinte sunt în afara vocabularului și nu pot fi prezise de către modelul de limbă. Cuvintele din afara vocabularului pot face sarcina de evaluare a unui model de limbă mai dificilă, deoarece perplexitatea lor este infinită, iar acest factor nu poate fi adunat la factorul de perplexitate ale celorlaltor n-grame în scopul obținerii perplexității întregii secvențe de cuvinte.

După cum a fost menționat, sarcina de recunoaștere a vorbirii continue este foarte dificilă deoarece pronunție oricărei secventțe de cuvinte este iminentă. În aceste condiții modelele de limbă statistice de tip n-gram sunt foarte potrivite. Dar, în condițiile în care se prezintă restricții cu privire la vocabularul de cuvinte și setul de succesiuni valide de cuvinte, modelul de tip n-gram nu poate fi folosit în vederea obținerii unor performanțe foarte bune. Succesiunile valide și probabilitățile de pariție pot fi specificate în mod direct folosind un model de limbă cu stări finite. Gramatica cu stări finite poate fi reprezentată ca un graf unde nodurile pot reprezenta cuvinte din vocabular, iar tranzițiile pot fi ilustrate ca arce ale grafului. Astfel de modele de limbă specifică în mod explicit toate secvențele de cuvinte permise de gramatica sarcinii de recunoaștere.

Pentru a ilustra modul de construcție al unei astfel de gramatici, vom lua exemplul recunoașterii de cifre, în care vocabularul este compus doar de către cifrele de la 0 la 9. Definirea grafului va începe cu definirea nodurilor de intrare și ieșire. Deoarece trecerea prin aceste noduri nu corespunde cu afișarea propriu-zisă a unui cuvânt vor fi notate cu N(null). Se vor crea noduri specifice pentru fiecare cuvânt și legarea acestora atât de nodurile de intrare cât și de cele de ieșire. Sarcina aceasta descrie recunoașterea pronunției unei singure cifre. Pentru recunoașterea unei secvențe de cifre, se va adăuga o tranziție înapoi (de la nodul de ieșire către cel de intrare). Astfel sistemul poate fi descris ca un sistem cursiv ce poate recunoaște secvențe de cifre. Importantp este adăugarea a două noduri de tip null, pentru că cele precedente (intrare/ieșire) nu mai au caracterul unor noduri I/O propriu-zise (acestea conțin și arce).

[9] Research and Development Project in Spoken Language Technology – Horia Cucu

Figura următoare ilustrează grafic gramatica cu stări finite descrise mai sus ce poate îndeplini sarcina de recunoaștere. Putem observa că modelul este acum format din 14 noduri, dintre care doar 10 presupun cuvinte ale limbii, celelalte putând fi descrise ca noduri de infrastructură, utilizate pentru intrari și ieșiri din graf, dar cel mai important pentru tranziția înapoi și recunoașterea aferentă a unei noi cifre.

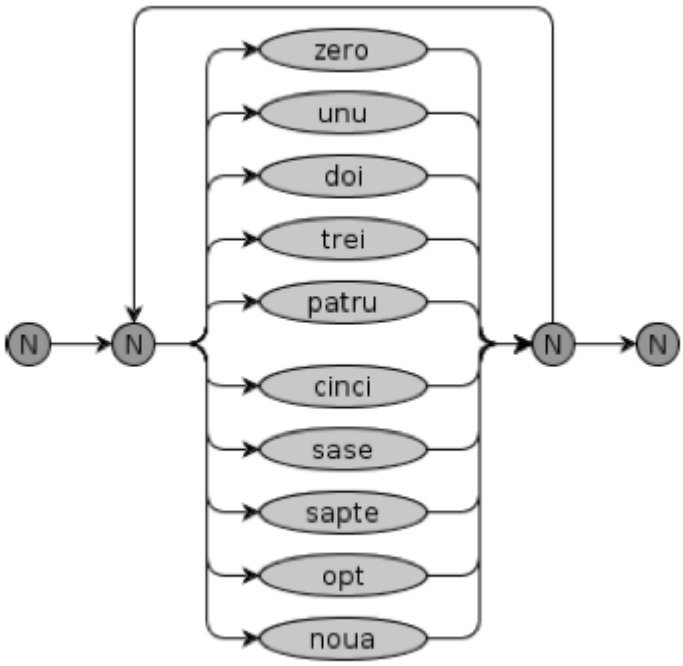

Fig 1. 5 Gramatica cu stări finite a sarcinii de recunoaștere de cifre

Acest sistem de recunoaștere a fost implementat în vederea atingerii obiectivului de familiarizare cu sistemul de recunoaștere de vorbire continuu bazat pe gramatică de tip FSG/JSFG.

Detalii aferente la implementare și la descrierea amănunțită a unei gramatici de tip Java Speech Grammar (JSFG) folosită în sistemul actual de recunoaștere a vorbirii directe prezente la posturile de radio sunt prezentate în amănunt la secțiunea de dezvoltare. În continuare vor fi descrise tehnologiile utilizate în implementare, iar apoi pașii propriu-ziși urmați în acest proces.

Capitolul 2. Tehnologii folosite în implementarea sistemului de recunoaștere automată a vorbirii

2.1 Introducere în limbajul de programare Java (dezvoltare, principii, performanțe, garbage colector)

Java este un limbaj de programare de nivel înalt bazat pe concurență, orientat-obiect ce a fost special conceput pentru a avea cât mai puține dependențe posibile. Acest limbaj de programare este destinat pentru a permite dezvoltatorilor de aplicații să scrie programul o singura dată, acesta putând fi rulat oriunde ("write once, run anywhere", WORA). Acest lucru înseamnă că rularea este independentă de platformă odată cu compilarea acesteia. Aplicațiile Java sunt compilate bytecode (fișiere clasă) și pot rula pe orice masină virtuală Java. Acest limbaj de programare este unul dintre cele mai populare limbaje utilizate la momentul actual, în special pentru aplicații web client-server, cu un raport de peste 9 milioane de dezvoltatori curenți înregistrati. Java a fost inițial dezvoltat de James Gosling la Sun Microsystems (firmă ce între timp a fuzionat cu Oracle Corporation) și lansat în anul 1995 ca o componentă esențială a platformei Java Sun Microsystems. Limbajul de programare derivă în mod direct din limbajele C și C++, dar conține mai puține facilități de nivel inferior (low-level) comparativ cu aceste limbaje de programare.[10]

Versiunile originale și implementările referință ale compilatoarelor Java, ale mașinii virtuale Java și ale librăriilor aferente au fost dezvoltate începând cu anul 1991 și lansate în premieră în anul 1995. Din mai 2007, în conformitate cu specificațiile Java Comunity Process, Sun a relicențiat marea majoritate a tehnologiilor Java sub GNU General Public Licence. Alte părți au dezvoltat implementări alternative ale acestor tehnologii Sun, cum ar fi GNU Compiler for Java (compilator bytecode), GNU Classpath (un pachet de librării standard) sau IcedTea-Web (un pachet suplimentar destinat browser-ului pentru aplicații de tip applet).

James Gosling, Mike Sheridan, și Patrick Naughton au inițiat proiectul limbajul Java în iunie 1991. Java a fost proiectat inițial pentru televiziunea interactivă, dar a fost mult prea avansat pentru industria de televiziune digitală la momentul respectiv. Limbajul a fost numit inițial Oak după un stejar care se afla în afara biroului lui Gosling; el a mers cu numele Green mai târziu, apoi redenumit Java, de la cafeaua Java. Gosling a propus să pună în aplicare o mașină virtuală și un limbaj familiar C / C ++ ca stil de notație.

Sun Microsystems a lansat prima versiune publică, Java 1.0 în 1995 . Acesta a promis implementarea unui limbaj "Write Once, Run Anywhere" (WORA), oferind o rulare fără costuri pe platformele populare. Destul de sigur și oferind posibilități de configurare avansate, această versiune a oferit restricții pentru fișiere și rețea. Majoritatea browserelor web au inclus în curând posibilitatea de a rula applet-uri Java în pagini web, astfel încât Java a devenit în scurt timp foarte popular. Odată cu apariția Java 2 (lansat inițial ca J2SE 1.2 în decembrie 1998-1999), această versiune a conținut mai multe elemente ușor configurabile pentru diferite tipuri de platforme. De

[10] http://en.wikipedia.org/wiki/Java_(programming_language)

exemplu, J2EE orientat către aplicații de întreprindere și versiunea J2ME pentru aplicatii mobile (Mobile Java). J2SE este acronimul pentru versiunea standard (Standard Edition). În 2006, în scopuri de marketing, Sun a redenumit noi versiunii J2 ca Java EE, Java ME, și respectiv Java SE.

În 1997, Sun Microsystems a abordat standardele ISO / IEC JTC1. La un moment dat, Sun a făcut cele mai multe din implementările sale Java disponibile în mod gratuit, în pofida statutului lor de software-ul proprietar. Sun a generat venituri din Java prin vânzarea de licențe pentru produse specializate, cum ar fi Enterprise System Java. Sun se distinge prin Development Kit Software (SDK) și Runtime Environment (JRE) (un subset SDK); distincția primară implică lipsa compilatorului, a programelor utilitare, sau a fișierelor antet.

La 13 noiembrie 2006, Sun a lansat o versiune avansată Java ca software cu sursă liberă și deschisă, (FOSS), în conformitate cu termenii GNU General Public License (GPL). La 8 mai 2007, Sun a terminat procesul, codul de bază Java fiind disponibil sub termeni de distribuire gratuită (software / open-source), exceptând o mică parte de cod asupra căruia Sun nu deține drepturile de autor.

După achiziționarea Oracle Corporation a Sun Microsystems în perioada 2009-2010, Oracle s-a descris ca "administrator al tehnologiei Java cu un angajament neobosit pentru promovarea unei comunități de participare și transparență ". [1] Acest lucru nu a împiedicat Oracle de la depunerea unui proces împotriva Google, la scurt timp, pentru utilizarea Java în Android SDK. Software-ul Java ruleaza pe orice platformă, de la laptop-uri la centre de date, console de jocuri și supercalculatoare științifice. Exista 930 milioane de download-uri Java Runtime Environment în fiecare an și 3 miliarde de telefoane mobile rulează cu ajutorul Java. La 2 aprilie 2010, James Gosling a demisionat de la Oracle.[11]

Crearea limbajului de programare Java a presupus respectarea a cinci obiective principale :

1. Ar trebui să fie "simplu, orientat-obiect și familiar" .

2. Ar trebui să fie "robust și sigur" .

3. Ar trebui să fie "independent de platformă și portabil"

4. Acesta ar trebui să fie executat cu " înaltă performanță"

5. Acesta ar trebui să fie "interpretat, bazat pe threaduri, și dinamic"[12]

Programele scrise în Java au reputația de a fi mai lente și necesită mai multă memorie decât cele scrise în C ++. Cu toate acestea, viteza de executie a programelor Java a crescut semnificativ odată cu introducerea de compilare Just-in-time în 1997/1998 pentru Java 1.1. Adăugarea de caracteristici de limbă suportă analiza de cod mai bine (cum ar fi clase interioare, clasa StringBuilder, afirmații opționale, etc), și optimizări în mașina virtuală Java în sine, cum ar fi de HotSpot-uri fac parte implicit din JVM Sun începând cu anul 2000.

[11] https://www.java.com/en/about/
[12] "The Java Language Environment". *1.2 Design Goals of the Java™ Programming Language*. Oracle.

Unele platforme oferă suport hardware direct pentru Java; există microcontrolere care pot rula Java în hardware în loc de un software Java virtual machine; procesoare ARM pot avea de asemena suport hardware pentru executarea Java bytecode, prin opțiunea lor Jazelle.

Java folosește un colector automat de gunoi pentru a gestiona memoria în ciclul de viață al obiectului. Programatorul determină atunci când sunt create obiecte, iar Java este responsabil pentru recuperarea memoriei odată ce obiectele nu mai sunt în uz. Dacă nici o referire la un obiect nu rămâne, memoria devine eligibilă pentru a fi eliberată în mod automat de către colectorul de gunoi. Fenomene similare cu o scurgere de memorie pot să apară în cazul în care codul unui programator deține o trimitere la un obiect care nu mai este necesar, de obicei, atunci când obiectele care nu mai sunt necesare sunt depozitate în containere care sunt încă în uz. În cazul în care sunt numite metode pentru un obiect inexistent, o "excepție pointer nul" este aruncată.[13]

Una dintre ideile din spatele modelului automat de gestionare a memoriei Java este că programatorii pot fi scutiți de sarcina de a trebui să efectueze gestionarea memoriei manual. În unele limbaje, memoria pentru crearea de obiecte este alocată implicit pe stivă, sau împărțită în mod explicit și dezalocată din spațiul heap. În cazul din urmă responsabilitatea de gestionare a memoriei îi revine programatorului. În cazul în care programul nu dezaloca un obiect, memoria este irosită și utilizată necorespunzător. În cazul în care programul încearcă să acceseze o parte din memorie ce a fost deja dezafectată, rezultatul este nedefinit și dificil de prezis, iar programul este probabil să devină instabil. Acest lucru poate fi remediat parțial prin utilizarea indicilor inteligenți, dar aceștia se adaugă deasupra header-ului și denotă o complexitate crescută. Colectarea gunoiului nu împiedică pierderi de memorie "logice", adică cele în care memoria este încă referire, dar nu a fost folosită niciodată.

Optimizarea memoriei prin înlăturarea obiectelor fără referință se poate întâmpla în orice moment. În mod ideal, aceasta va avea loc atunci când un program este inactiv. Aceasta este garantat declanșată în cazul în care există memorie liberă insuficientă în spațiul heap pentru a aloca un nou obiect; aceast caz poate provoca un program să se blocheze pentru moment. Managementul memoriei explicite nu este posibil în Java.

Java nu are suport pentru pointer aritmetic ca în stil C / C ++, unde adresele de obiecte și numerele întregi fără semn (de obicei numere întregi lungi) pot fi folosite alternativ. Acest lucru permite colectorul de gunoi să mute obiecte referite și asigură siguranța și securitatea.

Ca și în C ++ și alte limbaje orientate-obiect, variabilele de tip primitiv în Java nu sunt obiecte. Valorile de tip primitiv sunt fie stocate direct în domenii (de obiecte) sau pe stivă (pentru metode), mai degrabă decât în spațiul heap, așa cum este de obicei valabil pentru obiecte. Aceasta a fost o decizie de design Java pentru motive de performanță. Din acest motiv, Java nu a fost considerată a fi un limbaj de programare orientat pe obiect pur.[14]

Java conține mai multe tipuri de colectori de gunoi. În mod implicit, HotSpot folosește paralel colectorul de gunoi de tip baleiaj. Cu toate acestea, există, de asemenea, o serie de alte colectoare de gunoi care pot fi utilizate pentru a gestiona spațiul heap.

[13] http://en.wikipedia.org/wiki/Garbage_collection_(computer_science)
[14] http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

Se poate afirma că inima platformei Java o reprezintă conceptul de "mașină virtuală", care execută programele compilate bytecode. Aceste programe bytecode sunt aceleași, indiferent de sistemul hardware sau sistemul de operare ce le executată. Există un compilator JIT (Just In Time) în Java Virtual Machine, sau JVM.JIT ce traduce codul bytecode Java în instrucțiuni procesor nativ la momentul rulării și memorează în memoria cache codul nativ în timpul execuției. [15]

Utilizarea bytecode ca un limbaj intermediar permite programelor Java să ruleze pe orice platformă ce are o mașină virtuală disponibilă. Utilizarea unui compilator JIT denotă faptul că aplicațiile Java, după o scurtă întârziere în timpul de încărcare și după ce au efectuat compilarea codului, au tendința de a rula la fel de repede ca programe autohtone.

Deși programele Java sunt cross-platform sau independente de platformă, codul Java Virtual Machines (JVM) ce executa aceste programe nu este. Fiecare platformă de operare acceptată are propria JVM.

În cele mai multe sisteme de operare moderne, un volum mare de cod reutilizabil este furnizat pentru a simplifica sarcina programatorului. Acest cod este de obicei furnizat ca un set de biblioteci încărcate dinamic la care aplicațiile pot apela în timpul rulării. Deoarece platforma Java nu depinde de nici un sistem de operare specific, aplicațiile nu se poate baza pe oricare din bibliotecile sistemului de operare pre-existente. În schimb, platforma Java oferă un set cuprinzător de biblioteci de clasă standard care conțin o mare parte din funcțiile reutilizabile frecvent întâlnite în sistemele de operare moderne. Cea mai mare parte a bibliotecii sistem este, de asemenea, scrisă în Java. De exemplu, biblioteca Swing, construiește interfața cu utilizatorul și se ocupă de interacțiunea cu acesta, eliminând multe diferențe subtile între modul în care diferitele platforme se ocupă de componente similare.

Bibliotecile de clasă Java servesc trei scopuri în platforma Java. În primul rând, ca și alte biblioteci de cod standard, bibliotecile Java oferă programatorului un set de funcții bine-cunoscute pentru a efectua sarcini comune, cum ar fi menținerea listelor de articole sau efectuarea de parsing de șiruri complexe. În al doilea rând, bibliotecile de clasă oferă o interfață abstractă pentru sarcini ce ar depinde în mod normal în mare măsură de sistemul hardware și cel de operare. Sarcini cum ar fi accesul la rețea și accesul la fișiere sunt adesea puternic interconectate cu implementările distinctive ale fiecărei platforme. Bibliotecile java.net și java.io pun în aplicare un strat de abstractizare în cod nativ OS, apoi oferă o interfață standard aplicațiilor Java pentru a efectua aceste sarcini. În cele din urmă, atunci când unele platforme care stau la bază nu au suport pentru toate caracteristicile unei aplicații Java, bibliotecile de clase de lucru pot lua locul cu grație componentelor absente, fie prin emulare pentru a oferi un înlocuitor, sau cel puțin prin furnizarea unui mod consistent de verificare a prezenței unei caracteristici specifice.[16]

Deoarece programele scrise în limbajul Java rulează pe o mașină virtuală, se execută oarecum lent în comparație cu alte programe. Cu toate acestea, este puțin probabil ca efectul să fie resimțit pe calculatoarele foarte rapide din ziua de astăzi .

O problemă mai mare o reprezintă faptul că programele nu funcționează întotdeauna corect, chiar dacă acestea sunt scrise corect, pentru că o mașină virtuală Java poate fi scrisă incorect.

[15] http://searchsoa.techtarget.com/definition/Java-virtual-machine
[16] http://www.cl.cam.ac.uk/teaching/1112/OOProg/Files/OOPLecture8.pdf

Sarcina de a scrie un program perfect este dificilă, mai ales dacă acesta este la fel de mare ca o mașină virtuală, astfel încât programele scrise în Java au tendința de a suferi de ușor diferite probleme, în funcție de platformele pe care mașinile virtuale Java rulează. [17]

2.2 Librăria de recunoaștere a vorbirii CMU Sphinx4[18]

Sphinx-4 este un sistem de recunoaștere a discursului continuu stat-of-the-art, independent de vorbitor. Sistemul de recunoaștere este scris în întregime în limbajul de programare Java.

Acesta a fost creat printr-o colaborare între grupul Sfinx de la Carnegie Mellon University, Sun Microsystems Laboratories, Mitsubishi Research Labs electric (Merl), și Hewlett Packard (HP), cu

contribuții de la Universitatea California Santa Cruz (UCSC) și Institutul de Tehnologie din Massachusetts (MIT).

Designul Sphinx-4 se bazează pe modelele de bază obținute din proiectarea anterioară de sisteme de recunoaștere, precum și noi cerințe bazate pe cercetări în derulare. Implementările CMU-Sphinx sunt toate disponibile gratuit (open source).

Odată cu lansarea versiunii 1.0 beta, este disponibil arborele sursă Sphinx-4 împreună cu mai multe modele acustice și lingvistice capabile de rezolvare a unei varietați de sarcini, de la recunoaștere simplă a cifrelor la recunoașterea de tip n-Gram.

Deoarece este scris în întregime în limbajul de programare Java, Sphinx-4 poate rula pe o varietate de platforme, fără a necesita o compilare specială sau alte modificări. Acest sistem a fost testat pe

următoarele platforme cu succes: Solaris 9, platforma SPARC, Mac OS X 10.3.5, RedHat 9.0, Fedora Core 10 și distribuția de sisteme de operare oferită de Microsoft (Windows).

2.3 Noțiuni generale de arhitectură a unui sistem Sphinx4[19]

Sistemul de recunoaștere Sphinx-4 încorporează mai multe strategii de design, ce nu au fost utilizate în decodoare convenționale de vocabular mare bazate pe HMM. Unele aspecte legate de proiectare includ grafuri de construcție pentru decodare paralelă pe mai multe nivele, încorporarea unui algoritm de căutare generalizată care subsumează decodarea Viterbi și, în cazuri speciale, design de grafuri HMM pentru gramatici și modele lingvistice ce conțin multiple formate standard (comutarea de la structura de căutare plată la structura de căutare de tip arborescentă), etc.

Arhitectura Sphinx-4 a fost proiectată cu un grad ridicat de modularitate. Figura 2.1 prezintă arhitectura generală a sistemului. Chiar și în cadrul fiecărui modul prezentat în figura 2.1, codul este extrem de modular cu funcții ușor de înlocuit.

[17] Jelovic, Dejan. "Why Java Will Always Be Slower than C++". www.jelovic.com
[18] http://cmusphinx.sourceforge.net/wiki/#cmusphinx_wiki
[19] http://www.merl.com/publications/docs/TR2003-110.pdf

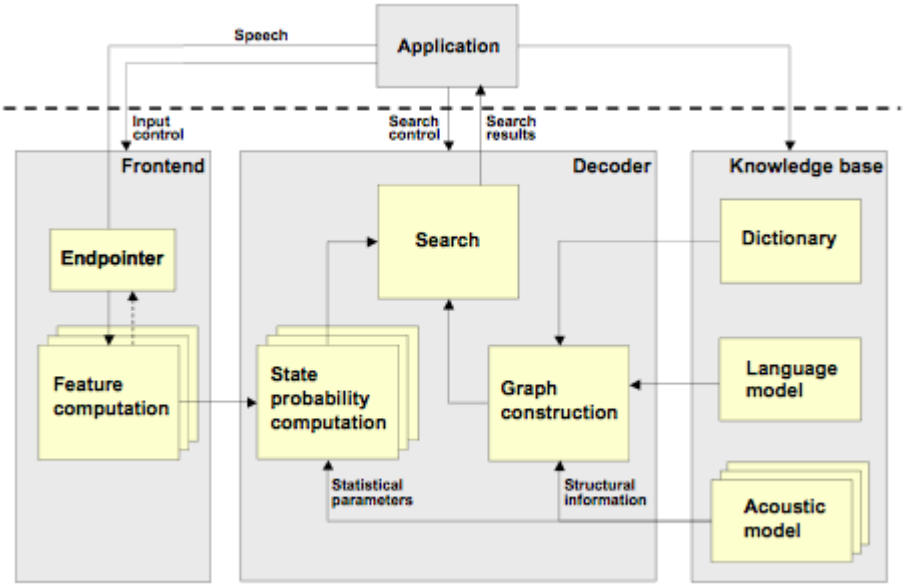

Fig 2. 1 Arhitectura generală a unui sistem de recunoaștere Sphinx4

Se pot observa trei blocuri principale în proiectare, controlabile de către orice aplicație externă: *frontend*, *decodor* și *baza de cunoștințe* (KB). Modulul *frontend* permite achiziția semnalului ce conține vorbire și se ocupă de parametrizare . În cadrul acestuia, modulul *endpointer* poate returna în final semnalul ce conține vorbire, sau vectorii de caracteristici calculați pornind de la semnalul de intrare. Blocul *decodor* realizează recunoașterea efectivă. Acesta este format dintr-un modul ce construiește grafuri, modul ce poate traduce orice tip de model de limbaj standard furnizat de KB. Aplicația poate accesa informațiile de la fiecare nod al grafului pentru a obține rezultate de căutare pe diferite niveluri (ex: partiționări la nivel de cuvânt sau segmentări). Aplicația poate controla, de asemenea, nivelul cu care fluxurile de caracteristici paralele sunt combinate în timpul căutării și modul în care este tratat fiecare flux de informații. Modulul de căutare necesită probabilitatea de apariție a oricărui vector caracteristic curent. Probabilitățile sunt calculate de către modulul de calcul separat. Calcularea scorului de traducere este, prin urmare, o sarcină ce se efectuează ori de câte ori modulul de căutare comunică cu modulul de notare. În Sphinx-4, modulul de construcție al grafului este, de asemenea, numit lingvist.

2.4 Jax-RS, API Java pentru dezvoltarea de servicii RESTful

JAX-RS este un API al limbajului de programare Java, care oferă suport în crearea de servicii web în funcție de model arhitectural Representational State Transfer (REST). JAX-RS folosește adnotări, introduse în Java SE 5, pentru a simplifica dezvoltarea și implementarea de clienti de servicii web.

De la versiunea 1.1 la, JAX-RS este o parte oficial din Java EE 6. O caracteristică remarcabilă este că nici o configurație nu este necesară pentru utilizața JAX-RS. Pentru medii non-Java EE 6 este necesară o intrare în descriptorul de implementare web.xml.[20]

2.5 Java Jersey

Serviciul Jersey RESTful Web Services este un cadru open source, pentru dezvoltarea de servicii Web RESTful în Java, care oferă suport pentru JAX-RS și servește ca o implementare de referință a acestuia.

Framework-ul Jersey este mai mult decât implementarea JAX-RS de referință. Jersey oferă un API propriu, care extinde setul de instrumente JAX-RS cu caracteristici suplimentare și utilități pentru a simplifica și mai mult serviciul RESTful și dezvoltarea de clienți. Jersey expune, de asemenea, numeroase extensii SPI, astfel încât dezvoltatorii pot extinde Jersey pentru a se potrivi cel mai bine nevoilor lor.[21]

2.6 Maven

Maven este un instrument ce își propune automatizarea dependențelor unui proiect și este utilizat în principal pentru proiectele scrise în limbajul Java. Maven abordează două aspecte ale software-ului dezvoltat: În primul rând, se descrie modul în care este construit software-ul, iar în al doilea rând, se descriu dependențele sale. Un fișier XML descrie modul de construcție al unui proiect software, dependențele sale de alte module externe, ordinea construirii, directoarele și plug-in-urile necesare. Acesta este dotat cu directive predefinite pentru efectuarea anumitor sarcini, cum ar fi compilarea codului și împachetarea acestuia. Maven descarcă dinamic biblioteci Java și plug-in-uri de la unul sau mai multe registre, cum ar fi Maven 2 Central Repository, și le stochează într-o memorie cache locală. Acest cache local de artefacte descărcate poate fi, de asemenea, actualizat cu artefacte create de proiectele locale. Registrele publice pot fi, de asemenea, actualizate.

Proiectul Maven este găzduit de către Apache Software Foundation și este construit folosind o arhitectură bazată pe plug-in. Teoretic, acest lucru ar permite oricui să scrie plugin-uri pentru a interfera cu uneltele de compilare (compilatoare, unelte pentru testare, etc) în orice alt limbaj de programare. În realitate, sprijinul și utilizarea în alte scopuri decât limbajul Java a fost minimă. [22]

2.7 Protocolul HTTP

Hypertext Transfer Protocol (HTTP) este un protocol pentru aplicații distribuite, sisteme de colaborare sau informare hypermedia. HTTP este fundamentul comunicației de date pentru World Wide Web.

[20] http://en.wikipedia.org/wiki/Java_API_for_RESTful_Web_Services
[21] https://jersey.java.net
[22] http://maven.apache.org/index.html

Hypertext este un text structurat care foloseste legături logice (hyperlink-uri) între noduri ce conțin text. HTTP este protocolul de schimb sau Hypertext Transfer.

Dezvoltarea standardelor HTTP a fost coordonată de către Internet Engineering Task Force (IETF) și World Wide Web Consortium (W3C).

HTTP funcționează ca protocol cerere-răspuns în modelul de calcul client-server. Un browser web, de exemplu, ar putea fi clientul și o aplicație care rulează pe un calculator ce găzduiește un website web poate fi server-ul. Clientul trimite un mesaj cerere HTTP către server. Serverul, ce poate oferi diferite resurse(fișierele HTML sau alte tipuri de conținut), sau poate executa alte funcții în numele clientului, returnează un mesaj de tip răspuns către client. Răspunsul conține informații de stare despre completarea cererii și poate conține, de asemenea, conținutul solicitat în corpul său mesaj.

Un browser web este un exemplu de un agent utilizator (UA). Alte tipuri de agent utilizator includ software-uri de indexare utilizate de către furnizorii de căutare (crawlerele web), browsere de voce, aplicații mobile, și alte software-uri care accesează, consumă, sau afișează conținut web.

HTTP este conceput pentru a permite elementelor intermediare de rețea să îmbunătățească sau să permită comunicarea dintre clienți și server.

HTTP este un protocol de strat aplicație conceput în cadrul Internet Protocol Suite. Definiția sa presupune existența unui protocol de strat transport(de obicei este utilizat Transmission Control Protocol (TCP)). Cu toate acestea HTTP poate utiliza și protocoale nesigure, cum ar fi User Datagram Protocol (UDP).[23]

2.9 SSE - Server Sent Events. Chunked transfer encoding

Evenimentele trimise de către server (SSE) reprezintă o tehnologie în care un browser primește actualizări automate de la un server printr-o conexiune HTTP. EventSource API este standardizat, ca parte din HTML5 de către W3C.

SSE este un standard care descrie modul în care serverele pot iniția transmiterea datelor către clienți odată ce conexiune inițială a fost stabilită. Ele sunt de obicei folosite pentru a trimite actualizări de mesaje sau fluxuri de date continue la un client browser și concepute pentru a spori nativ, streaming-ul cross-browser prin intermediul unui API JavaScript numit EventSource. (un client solicită un anumit URL, în scopul de a primi un flux de evenimente)

Chunked transfer encoding este un mecanism de transfer de date în versiunea 1.1 a Hypertext Transfer Protocol (HTTP), în care este trimis un flux de date într-o serie de "bucăți". Se folosește antetul HTTP Transfer-Encoding în loc de antetul Content-Length. Deoarece antetul Content-Length nu este folosit, expeditorul nu are nevoie să știe lungimea conținutului înainte de a începe transmiterea unui răspuns la receptor. Expeditorii pot începe transmiterea de conținut generat dinamic înainte de a cunoaște dimensiunea totală a acelui conținut.

[23] http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

Dimensiunea fiecărei bucăți este trimisă chiar înainte de aceasta sine, astfel încât receptorul să poată spune când a terminat de recepționat datele. Transferul de date se termină cu o bucată finală de lungime zero.

Introducerea mecanismului Chunked transfer encoding în HTTP 1.1 a furnizat un număr de beneficii:

Chunked transfer encoding permite unui server să mențină o conexiune HTTP persistentă c. În acest caz, antetul HTTP Content-Length nu pot fi folosit pentru delimitarea conținutului și a următoarei/următorului cereri/răspuns HTTP, deoarece dimensiunea de conținut este încă necunoscută. Codarea chunked este avantajoasă deoarece nu este necesară generarea întregului conținut înainte de a scrie antetul; permite redarea de conținut direct ca bucăți și semnalizează explicit sfârșitul conținutului, făcând legătura disponibilă pentru următoarea/următorul cerere / răspuns HTTP.

Codare chunked permite expeditorului să trimită câmpuri suplimentare (antet) după corpul mesajului. Acest lucru este important în cazurile în care valorile de câmp nu pot fi cunoscute înainte de producerea conținutului, cum ar fi atunci când conținutul mesajului trebuie semnat digital. Fără codare chunked, expeditorul va trebui să memoreze temporar conținutul până când a fost complet, în scopul de a calcula o valoare de câmp și a o trimiteînainte de conținutul.

Aplicabilitate :

Pentru versiunea 1.1 a protocolului HTTP, mecanismul de transfer chunked este considerat a fi întotdeauna acceptat, chiar dacă nu este enumerat în TE (codificare de transfer) câmpul antet al cererii. Această metodă de codificare de transfer permite, de asemenea, câmpurilor suplimentare antet să fie trimise după ultima bucată trimisă, în cazul în care clientul a specificat parametrul *trail*, ca un argument al câmpului TE. Serverul poate decide, de asemenea, retrimiterea de entități suplimentare de tip *trail* chiar dacă clientul nu a specificat opțiunea aceasta în cererea facută, dar numai în cazul în care metadatele sunt opționale . Ori de câte ori sunt folosite remorci, serverul ar trebui să enumere numele lor în domeniul antet Trailer.[24][25]

[24] http://en.wikipedia.org/wiki/Server-sent_events
[25] http://www.w3schools.com/html/html5_serversentevents.asp

Capitolul 3. Sistemul de recunoaștere Radio Transcriber

Arhitectura sistemului de recunoaștere a vorbirii continue la radio presupune dezvoltarea unui serviciu de tip RESTful în Java cu ajutorul tehnologiilor web (Glasfish Server) și Jersey (server-sent events). S-a impus dezvoltarea unui client cu ajutorul tehnologiilor web de actualitate (html5, javascript, bootstrap) cu ajutorul căruia partea de server a fost testată. În figura 3.1 este explicată arhitectura generală a sistemului și sunt descrișii pașii urmăriți în rularea aplicației (flow diagram).

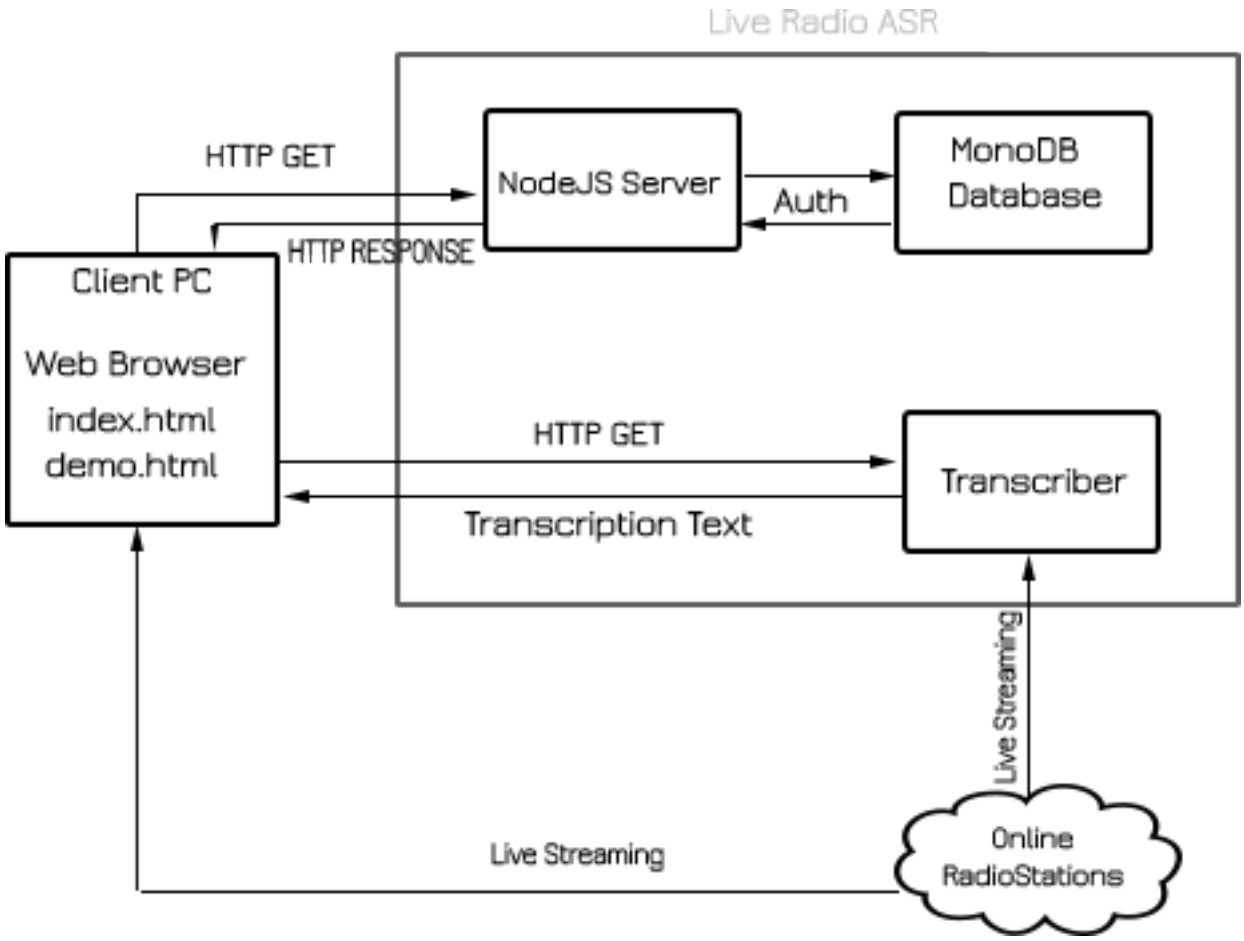

Fig 3. 1 Arhitectura generală a serviciului web de transcriere automată a știrilor difuzate la radio

Componenta principală a acestui sistem este server-ul Java ce efectuează continuu transcrieri pentru diferite posturi de radio și așteaptă conectarea clienților.

3.1 Serviciul de autentificare NodeJS-MongoDB

Primul pas este autentificarea realizată de către server-ul web NodeJS. Stocarea informațiilor secrete (username, parolă) se face cu ajutorul unei baze de date NoSQL MongoDB.

În cazul în care datele provenite de la utilizator sunt corecte, acesta poate accede în partea de prezentare a serviciului și în secțiunea demo.

Trebuie menționat că alegerea acestor tehnologii pentru autentificare și pentru management-ul rutelor http a fost ales ca studiu de caz, diferite abordări fiind de asemenea disponibile (autentificare direct la server-ul Java, etc.).

Serverul Node este construit pentru a procesa acțiuni conduse de evenimente și a le procesa în mod asincron având o manieră non-blocantă. Mai precis, de fiecare dată când un eveniment apare la server, în timpul de procesare al acestuia server-ul poate accepta și procesa alte evenimente, urmând să revină la primul atunci când procesul s-a încheiat prin intermediul unei funcții de apel (callback). Să luăm exemplul de upload al unui fișier foarte mare. Clientul lansează acest eveniment către server, este evident că timpul de upload al unui fișier de dimensiuni mari va fi proportional cu dimensiunea acestuia. Datorită caracterului non-blocant, pe tot parcursul procesării fișierului server-ul poate primi și procesa o serie de alte evenimente urmând să revină la sarcinile de după upload.

Acest mod de acționare asincron și non-blocant a făcut tehnologia NodeJS să fie un excelent material pentru studiul de caz.[26]

Bazele de date NoSQL cuprind o mare varietate de tehnologii de baze de date și au fost dezvoltate ca răspuns la creșterea volumui de date stocate despre utilizatori, obiecte și produse, frecvența cu care datele sunt procesate și nevoile actuale de performanță și procesare. Bazele de date relaționale nu au fost concepute pentru a putea rezista provocărilor de scalabilitate și agilitate din aplicațiile moderne și, de asemenea, nu au fost concepute pentru a putea utiliza metodele de stocare foarte ieftine și puterea de procesare actuală.

În comparație cu bazele de date relaționale, bazele de date NoSQL au o putere de scalabilitate superioară beneficiind de performanțe crescute. Acest lucru este datorat faptului că nu mai este nevoie de dfiniția unor relații standard. Acestea pot fi adaptate în funcție de necesitatea descrierii unui model de date. De asemenea, modelul de date poate realiza sarcini pentru care sistemele de baze de date relaționale nu au fost concepute dintre care putem aminti volumele mari de date structurate, semi-structurate sau nestructurate, iterații și filtrări foarte rapide și modul de programare stil obiect-orientat. Stocarea datelor cu ajutorul formatului JSON binary ocupă un spațiu mai mic comparative cu bazele de date SQL.

MongoDB ("enorm") este o bază de date orientată spre documente. Clasificată ca o bază de date NoSQL, MongoDB evită structura tradițională de baze de date relaționale bazată pe tabele în favoarea documentelor JSON, cu scheme dinamice (MongoDB numește acest format BSON json-binar), ceea ce face integrarea datelor în anumite tipuri de aplicații mai ușor și mai rapid.

În acest tip de baze de date datele sunt încapsulate după modelul unei scheme dinamice sub formă de colecții și documente, colecțiile neformalizând în nici un fel structura documentelor. Această flexibilitate facilitează maparea unui document ca o entiate sau un obiect. [27]

În cadrul aplicației curente, s-a impus crearea unui model Utilizator ce poate stoca informațiile necesare autentificării locale și cu diferite rețele de socializare. Figura 3.2 arată

[26] http://www.javaworld.com/article/2079190/scripting-jvm-languages/6-things-you-should-know-about-node-js.html
[27] http://www.mongodb.org/about/introduction/

modelul folosit pentru stocarea utilizatorilor. Reamintim aici că formatul acestui obiect este de tip JSON .

```
var userSchema = mongoose.Schema({

    local            : {
        email        : String,
        password     : String,
    },
    facebook         : {
        id           : String,
        token        : String,
        email        : String,
        name         : String
    },
    twitter          : {
        id           : String,
        token        : String,
        displayName  : String,
        username     : String
    },
    google           : {
        id           : String,
        token        : String,
        email        : String,
        name         : String
    }

});
```

Fig 3. 2 Modelul de date al utilizatorului.

Autentificarea cu ajutorul rețelelor de socializare este realizată cu ajutorul framework-ului OAuth 1.0, mai precis, utilizatorul primește un token și un id asociat cu contul rețelei de socializare cu care s-a efectuat autentificarea, aceste variabile fiind stocate în obiectul ce definește utilizatorul. Mecanismul de autentificare bazat pe token este dezvoltat în următorul mod : la autentificare, utilizatorului i se atribuie o variabilă de tip token ce are acces restricționat asupra datelor acestuia (timp de utilizare, informații cu caracter personal, etc.) Această metodă este preferată deoarece utilizatorul nu va expune aplicației date cu caracter de securitate : nume utilizator și parolă.

Pentru configurarea autentificării cu rețele sociale, a fost folosit un modul NodeJS denumit PassportJS. Pentru utilizarea acestui modul se impune configurarea diferitelor strategii de autentificare. În cazul acestui sistem, s-a impus configurarea strategiei de autentificare și înregistrare locală și strategiile specifice rețelelor de socializare.

Passport este conceput pentru a servi doar scopului de autentificare. Atunci când sunt scrise modulele, încapsularea datelor este cea mai importantă caracteristică astfel încât Passport deleagă toate celelalte funcționalități aplicației (server-ului). Această separație a conceptelor menține codul

curat, structurat și facilitează întreținerea. Aceste concepte au fost avantajele pentru care a fost ales acest modul.

Așa cum am spus, modulul Passport funcționează pe baza strategiilor de autentificare descrise de către dezvoltator. În continuare, figurile 3.3 și 3.4 descriu strategiile de autentificare dezvoltate pentru autentificare locală și autentificare cu ajutorul rețelei Google+.

```
// =========================================================================
// LOCAL LOGIN =============================================================
// =========================================================================
// we are using named strategies since we have one for login and one for signup
// by default, if there was no name, it would just be called 'local'

passport.use('local-login', new LocalStrategy({
    // by default, local strategy uses username and password, we will override with email
    usernameField : 'email',
    passwordField : 'password',
    passReqToCallback : true // allows us to pass back the entire request to the callback
},
function(req, email, password, done) { // callback with email and password from our form

    // find a user whose email is the same as the forms email
    // we are checking to see if the user trying to login already exists
    User.findOne({ 'local.email' :  email }, function(err, user) {
        // if there are any errors, return the error before anything else
        if (err)
            return done(err);

        // if no user is found, return the message
        if (!user)
            return done(null, false, req.flash('loginMessage', 'No user found.')); // req.flash

        // if the user is found but the password is wrong
        if (!user.validPassword(password))
            return done(null, false, req.flash('loginMessage', 'Oops! Wrong password.')); // cr

        // all is well, return successful user
        return done(null, user);
    });

}));
```

Fig 3. 3 Strategia de autentificare locală din cadrul modulului Passport

Așa cum putem observa, această strategie presupune căutarea în baza de date a datelor de autentificare provenite de la utilizator și returnarea răspunsului corespunzător. Deoarece funcția *passReqToCallback* are argumentul *true,* în cazul rezultatului pozitiv către server-ul Node se întoarce un obiect JSON ce definește user-ul.

```
// =========================================================================
// GOOGLE ==================================================================
// =========================================================================
passport.use(new GoogleStrategy({

    clientID        : configAuth.googleAuth.clientID,
    clientSecret    : configAuth.googleAuth.clientSecret,
    callbackURL     : configAuth.googleAuth.callbackURL,

},
function(token, refreshToken, profile, done) {

    // make the code asynchronous
    // User.findOne won't fire until we have all our data back from Google
    process.nextTick(function() {

        // try to find the user based on their google id
        User.findOne({ 'google.id' : profile.id }, function(err, user) {
            if (err)
                return done(err);

            if (user) {

                // if a user is found, log them in
                return done(null, user);
            } else {
                // if the user isnt in our database, create a new user
                var newUser          = new User();

                // set all of the relevant information
                newUser.google.id    = profile.id;
                newUser.google.token = token;
                newUser.google.name  = profile.displayName;
                newUser.google.email = profile.emails[0].value; // pull the first email

                // save the user
                newUser.save(function(err) {
                    if (err)
                        throw err;
                    return done(null, newUser);
                });
            }
        });
    });
```

Fig 3. 4 Strategia de autentificare Google din cadrul modulului Passport

Strategia de autentificare Google presupune căutarea unui user în baza de date după id-ul provenit de la această rețea și autentificarea acestuia. În cazul în care acest user nu a mai fost autentificat niciodată se va crea un nou user în baza de date cu informațiile provenite de la contul său social. Recursiv, această strategie a fost implementată pentru rețelele de socializare Facebook și Twitter. Pentru mai multe informații privind implementarea acestui modul și a strategiilor aferente putem accesa fișierul *passport.js* ce se găsește în folder-ul *config*. (*config/passport.js*).

Pentru serializarea user-ului este folosit mecanismul bazat pe sesiuni (la autentificare se deschide o nouă sesiune în browser-ul web al clientului, sesiune ce expiră la închiderea paginii sau a browser-ului).

Mecanismul de procesare al cererilor http provenite de la client a fost implementat cu ajutorul unui alt modul Node denumit *Expressjs*. Acest modul este destinat exclusiv procesării cererilor http și respectă standardele de încapsulare descrise mai sus. Figura 3.5 arată modul de

procesare al unei cereri http de tip GET log-in și al unei cereri de tip POST log-in (trimiterea către server a informațiilor de autentificare).

```
app.get('/login', function(req, res) {

    // render the page and pass in any flash data if it exists
    res.render('login.ejs', { message: req.flash('loginMessage') });
});

// process the login form
app.post('/login', passport.authenticate('local-login', {
    successRedirect : '/profile', // redirect to the secure profile section
    failureRedirect : '/login', // redirect back to the signup page if there is an error
    failureFlash : true // allow flash messages
}));
```

Fig 3. 5 Exemplu de procesare al cererilor prin intermediul modulului Expressjs

Putem observa că la cererea de obținere a paginii de log-in răspunsul server-ului este randarea paginii login, iar la postarea informațiilor de autentificare de către client, server-ul apelează modulul Passport descris mai sus și returnează pagina de profil în cazul în care autentificare a fost efectuată sau pagina de log-in împreună cu mesajul flash aferent (e-mail/parolă greșit/ă).

3.2. Dezvoltarea unui sistem de recunoaștere automată a cifrelor.

Așa cum am menționat în capitolul introductiv al acestui document, pentru familiarizarea cu tehnica de recunoaștere a vorbirii directe s-a impus construirea unui sistem demo de recunoaștere a secvențelor continue de dialog ce conțin cifre. Arhitectura generala a unui sistem de recunoaștere este prezentată în figura 1.2 . Analizând această figură putem deduce că procesul de recunoaștere a vorbirii presupune extragerea unor parametri vocali din semnal, iar apoi transcrierea propriu-zisă pe baza unor modele (acustic, fonetic, lingvistic). Aceste modele sunt dezvoltate în prealabil.

Modelul acustic estimează probabilitatea unui mesaj vorbit în contextul unei succesiuni de cuvinte. Pentru sistemele de recunoaștere a vorbirii state-of-the-art modelul acustic nu folosește cuvintele ca unități de baza deoarece fiecare sarcină de recunoaștere conține un vocabular independent pentru care nu există modele antrenate în prealabil și nici date de antrenare. De asemenea, numărul de cuvinte ce compun o limbă este foarte mare. În locul cuvintelor, sunt folosite unități acustice sub-lexicale (foneme, senone). Așadar, modelul acustic este format dintr-un set de foneme sau senone ce se conectează în timpul procesului de decodare formând modele pentru cuvinte și modele pentru succesiuni de cuvinte. Cu ajutorul modelelor sunt estimate probabilitățile ca msajul rostit de vorbitor să fie format dintr-o succesiune de cuvinte. Implementarea generală pentru modelele acustice cu ajutorul modelelor Markov ascunse (hidden Markov models HMM) este prezentată în capitolul 1.

Modelul de limbă estimeaza probabilitățile tuturor secvențelor de cuvinte din spațiul căutării. În general, rolul acestui model este de a estima probabilitatea ca o secvență de cuvinte să formeze o propoziție validă a limbii

Această probabilitate ajută modelul acustic în procesul de decizie.

Modelul fonetic are rolul de a conecta modelul acustic cu cel de limbă. Acesta poate lua forma unui dicționar de pronunție ce asociază fiecărui cuvânt una sau mai multe secvențe de foneme adecvate. reprezentând modul de pronunție al fiecărui cuvânt.

În figura 1.2 sunt ilustrate procesele implicate în dezvoltarea unui sistem de recunoaștere, dar și resursele necesare creării modelelor acustice, lingvistice și fonetice. Modelul acustic este construit cu ajutorul unui set de clipuri audio înregistrate asociate cu transcrierea textuală a mesajelor vorbite și a unui dicționar format din toate cuvintele din transcrierea textuală. Pentru sistemele de recunoaștere cu vocabular extins, modelele de limbă folosite sunt cele statistice, formate utilizând corpusuri de text, dar pentru acest proiect (recunoașterea cifrelor), deoarece vocabularul este unul redus, vom folosi modele de limbă de tip gramatică cu stări finite, pentru construcția acestora nefiind nevoie de resurse textuale.

Modul de construcție al gramaticii este ilustrat și descris în figura 1.5.

3.2.1 Înregistrarea unei baze de date de clipuri audio necesare construirii modelului acustic

Pentru construirea modelului acustic s-a recurs la înregistrarea unei baze de date populată cu 100 de clipuri audio înregistrate independent, pentru un vorbitor. Aceste clipuri audio au fost înregistrate în laborator și diferite cifre simple și compuse rostite într-un mod cât mai natural.

3.2.2 Crearea dicționarului fonetic și a listei de foneme

Dicționarul fonetic este definit ca un instrument lingvistic ce specifică modul în care sunt pronunțate cuvintele unei limbi. Acesta face corespondența dintre forma scrisă și forma fonetică a cuvintelor unei limbi. Forma fonetică a cuvintelor este succesiunea de foneme ce descriu un anumit cuvânt. Tabelul 3.1 ilustrează fonemele limbii române pentru vocale. Reamintim că fonemul reprezintă unitatea de sunet fundamentală a limbii române.

Fonemul			Exemple de cuvinte	
Tip	**Simbol IPA**	**Simbol intern**	**Forma scrisă**	**Forma fonetică**
vocale	a	a	sat	s a t
	e	e	mare	m a r e
	i	i	lift	l i f t
	o	o	loc	l o c
	u	u	şut	s1 u t
	ə	a1	gură	g u r a1
	ɨ	i2	între	i2 n t r e

Tabelul 3. 1 Fonemele pentru vocalele limbii române

Pentru un sistem de recunoaștere a vorbirii continue dicționarul are rolul de a lega modelul acustic cu cel de limbă. Acest dicționar trebuie să conțină toate cuvintele posibile pentru sarcina de recunoaștere. Având în vedere sarcina de recunoaștere a secvențelor audio ce conțin cifre, este evident că dicționarul va trebui să conțină numai cifrele limbii române (0-9).

Sistemul de recunoaștere dezvoltat cu ajutorul sistemului CMU Sphinx4 utilizează dicționare fonetice în format text conținând un cuvânt pe fiecare linie a fișierului. Transcrierea fonetică a acestora va fi pe aceeași linie. Fonemele vor fi separate de un spațiu. În continuare vor fi prezentați pașii urmăriți în creearea dicționarului fonetic pentru sarcina de recunoaștere a cifrelor.

Primul pas este crearea unui fișier de tip text cu extensia .dic ce va conține fiecare cifră alături de transcrierea sa fonetică . Exemplu : *zero z e r o unu u n u , etc.*

Manualul sistemului CMU Sphinx recomandă utilizarea de foneme specifice pentru fiecare cuvînt. Așadar, fiecare cuvânt va trebui modelat diferit pentru un același fonem X în funcție de poziția din cuvânt în care apare acesta. Fișierul dicționar va avea următoarea formă :
zero z_zero e_zero r_zero o_zero, unu u_unu1 n_unu u_unu2 , etc. Lista fonemelor a fost apoi ordonată alfabetic, adăugându-se și fonemul SIL ce reprezintă fonemul specific pentru momentele de liniște.

3.2.3 Antrenarea modelului acustic

Procesul de antrenare a modelului acustic a fost realizat în cadrul unui proiect CMU Sphinx ce conține următoarele resurse: setul de clipuri audio înregistrat anterior, transcrierea textuală corespunzătoare cuvintelor pronunțate și dicționarul fonetic construit construit conform informațiilor prezentate mai sus. Pentru rezultate bune în procesul de recunoaștere, vom folosi și un dicționar cu elemente acustice ce nu sunt foneme (liniște, muzică, râs, etc).

Înainte de antrenarea modelului acustic, a fost creat un fișier ce conține lista clipurilor audio folosite la antrenare și un fișier corespunzător cu transcrierea textuală a clipurilor. Pentru procesul de antrenare au fost folosite doar 80 de clipuri audio, celelalte 20 fiind folosite în procesul de

evaluare al sistemului, proces ce va fi descris într-o secțiune viitoare. Transcrierea textuală a acestor clipuri reprezintă textul folosit pentru înregistrare. Sistemul CMU Sphinx4 necesită un fișier de configurare ce specifică parametrii modelului acustic ce va fi antrenat și locația resurselor. Crearea acestui fișier a presupus identificarea pe disc a căilor către resurse și alăturarea acestor căi variabilelor globale specifice CMU Spinx. Exemplul următor specifica configurarea dicționarului: *$CFG_DICTIONARY = "$CFG_LIST_DIR/rodigits.dic"*. Pasul următor a fost generarea de fișiere cu coeficienți MFCC corespunzători clipurilor audio înregistrate. Utilizând sistemul CMU Sphinx, vor fi create 80 de fișiere cu coeficienți cepstrali asociate fiecărui clip audio.

În final, aceste resurse generate sunt folosite pentru antrenarea modelului acustic.

Observație : În acest moment sistemul de recunoaștere este dependent de vorbitor (antrenarea sistemului a fost făcută cu înregistrări provenite numai de la un vorbitor).

3.2.4 Crearea modelului de limbă (gramatica)

Așa cum am menționat, deoarece sarcina de recunoaștere a cifrelor presupune un vocabular redus, sistemul de recunoaștere va folosi o gramatică cu stări finite (FSG). Acest model de gramatică este descris în capitolul 1 și ilustrat în figura 1.5 . Aceast model de gramatică presupune un model de tip graf, nodurile acestuia fiind reprezentate de cuvinte ale limbii, iar tranzițiile de arcele grafului. Acest model specifică explicit secvențiele de cuvinte permise de gramatica sarcinii de recunoaștere. Fiecărui arc îi poate fi asignat un cost specificând probabilitatea ca un cuvânt să fie precedat de un altul (probabilitatea unei secvențe de cuvinte). În sistemul CMU Sphinx, crearea gramaticii presupune crearea unui fișier cu extensia JSGF ce va avea următorul format :

#JSGF V1.0;

grammar rodigits;

*public <numbers> = (zero | unu | doi | trei | patru | cinci | şase | şapte | opt | nouă) * ;*

Acest fișier va fi transformat cu ajutorul utilitarului sphinx_jsgf2fsg din format JSGF în format FSG (finite state grammar) .

3.2.5 Evaluarea sistemului de recunoaștere

După configurarea celor trei componente necesare sistemului de recunoaștere (modelul acustic, modelul de limbă și modelul fonetic) se poate trece la decodarea clipurilor audio de evaluare și compararea transcrierii cu cea de referință. Pentru procesul de decodare se va folosi generarea de coeficienți cepstrali pentru cele 100 de clipuri înregistrate urmând ca acestea să fie decodate cu ajutorul utilitarului CMU Sphinx.

3.2.6 Interpretarea rezultatelor

Evaluarea sistemului de recunoaștere va fi efectuată comparând automat transcrierile textuale ale clipurilor de evaluare cu transcrierea de referință. Analiza se va face frază cu frază în două etape : fraza ipotetică și faza de referință sunt aliniate cu ajutorul unui algoritm ce urmărește minimizarea numărului de erori de transcriere, iar apoi erorile de recunoaștere la nivel de cuvânt (cuvinte inserate, substituite, șterse) se vor numpra. Criteriile de performanță standard pentru evaluarea sistemelor de recunoaștere a vorbirii continue vor fi astfel : rata de eroare la nivel de propoziție (SER - sentence error rate) și rata de eroare la nivel de cuvânt (WER - word error rate). Rata de eroare la nivel de cuvânt va fi calculată cu următoarea formulă :

$$WER[\%] = \frac{\#Erori\,de\,inserție + \#Erori\,de\,substituție + \#Erori\,de\,ștergere}{\#Cuvinte\,în\,transcrierea\,de\,referință} \times 100$$

Pentru sistemul curent dezvoltat, rezultatele înregistrate sunt prezentate în Tabelul 3.2 :

Sistem independent de context	
Wer : [%]	1.3 %

Tabelul 3. 2 Evaluarea sistemului de recunoaștere pentru un singur vorbitor, independent de context

3.2.7 Optimizarea sistemului acustic

Având în vedere cunoștințele prezentate în capitolul 1 cu referire la arhitectura unui model și alegerea unităților fonetice, în continuare va fi prezentat modul în care sistemul CMU Sphinx relizează antrenarea modelului acustic.

Standard, modelul acustic este construit cu unități de vorbire dependente de context de tip trifonem (unui fonem i se precizează vecinii din stânga și dreapta). Aceste trifoneme sunt implementate ca modele Markov ascunse cu 3 cu trei stări emisive, fiecare având o distribuție de probabilitate de ieșire implementată ca un GMM. Fiecare stare-model se numește senonă și poate fi comună mai multor trifoneme. Pentru sistemul CMU Sphinx, în mod implicit, numărul de senone este configurat la 200, iar numărul de desități de probabilitate pentru fiecare GMM este configurat la 8. Așadar, modelul acustic antrenat modelează cele câteva foneme specificate în dicționar utilizând modele fonetice dependente de context având 200 de senone, fiecare senonă folosind 8 densități Gaussiene de probabilitate pentru modelarea parametrilor acustici de ieșire. Procesul de antrenare CMU Sphinx urmează următorii pași în obșinerea modelului acustic :

1. se inițializează modelele fonetice independente de context (un model cu trei senone folosind o singură densitate gausiană pentru fiecare fonem)

2. antrenarea acestor modele fonetice (independente de context)

3. transformarea modelelor fonetice independente de context în modele fonetice dependente de context (se obține un model cu 3 senone și o singură densitate Gaussiană pentru fiecare trifonem în parte)

4. antrenarea noilor modele fonetice (independente de context)

5. legarea senonelor dependente de context pe baza similarităților dintre ele (se va obține câte un model cu câte trei senone, o singură densitate Gaussiană, pentru fiecare trifonem în parte, în total modelul acustic va avea 200 de senone).

6. antrenarea modelelor fonetice rezultate la pasul anterior

7. dublarea numărului de densitățâ Gaussiene pentru fiecare senonă (se va obține un model cu câte trei senone folosind două densități Gaussiene pentru fiecare trifonem în parte, în total 200 de senone)

8. antrenarea modelelor fonetice rezultate la pasul anterior

9. dublarea numărului de densităș](*) Gaussiene în cadrul fiecărei senone (se va obține un model cu câte trei senone folosind câte patru densități Gaussiene pentru fiecare trifonem în parte, în total 200 de senone)

10. antrenarea modelelor fonetice rezultate la pasul anterior

11. dublarea numărului de densități Gaussiene în cadrul fiecărei senone (se va obține un model cu câte trei senone folosind câte opt densități Gaussiene pentru fiecare trifonem în parte, în total 200 de senone)

Pentru evaluare s-a antrenat, de asemenea, un model acustic folosind doar 100 de senone.

Rezultatele decodării sunt prezentate în următorul tabel :

Nr. Gaussiene	WER[%] | 100 senone	WER[%] | 200 senone
1	2.4	7.1
2	2.4	11.3
4	1.2	10.8
8	0.8	12.4

Tabelul 3. 3 Evaluarea sistemului de recunoaștere pentru un singur vorbitor, dependent de context .

Din datele exprimate în tabel se observă că acuratețea unui sistem de recunoaștere crește odată cu creșterea numărului de densități Gaussiene, însă, pentru cazul sarcinii de recunoaștere a cifrelor, odată cu creșterea numărului de senone, această acuratețe scade. Figura 3.6 ilustrează grafic rezultatele prezentate în tabelul 3.3.

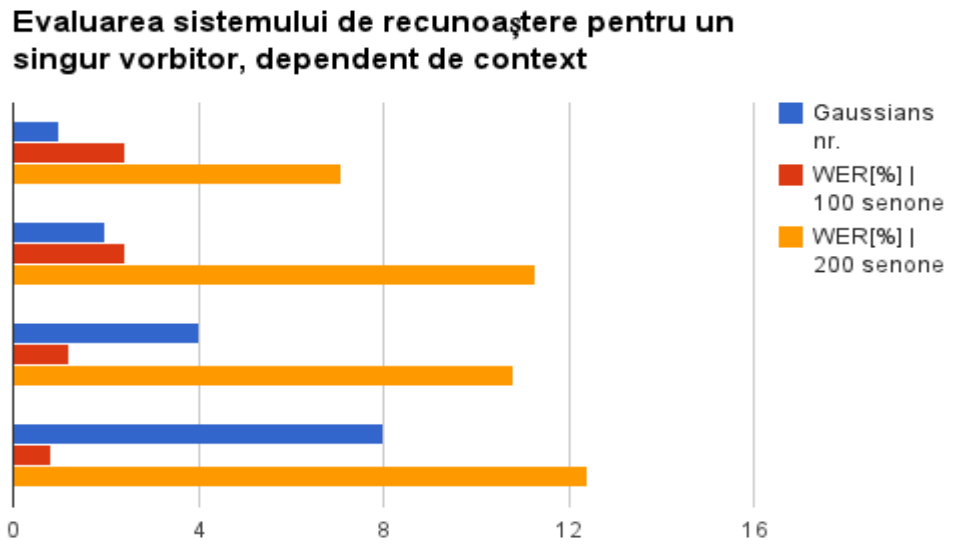

Fig 3. 6 Evaluarea sistemului de recunoaștere pentru un singur vorbitor, dependent de context.

3.2.8 Crearea unui sistem de recunoaștere independent de vorbitor

Deoarece sistemul de recunoaștere a cifrelor a fost antrenat doar cu datele provenite de la clipurile audio provenite de la un singur vorbitor, rezultatele în cazul decodării cu secvențe audio provenite de la diferiți vorbitori vor avea erori foarte mari.

Așadar, se impune antrenarea sistemului de recunoaștere cu clipuri audio provenite de la mai mulți vorbitori . Concret, sistemul de recunoaștere a fost antrenat ca în cazul unui vorbitor (caz prezentat mai sus), diferența constând în numărul clipurilor audio înregistrate, acestea provenind de la 17 vorbitori diferiți.

Observație : Clipurile audio înregistrate au fost aceleași astfel încât nu s-a impus schimbarea dicționarului fonetic sau a gramaticii necesare sarcinii date.

Rezultatele utilizării sistemului de recunoaștere antrenat cu date provenite de la 17 vorbitori diferiți sunt prezentate în continuare . Tabelul 3.4 prezintă evaluarea sistemului în cazul sistemului antrenat cu un singur vorbitor, dar testat și evaluat cu date de la 5 vorbitori diferiți. Se poate observa că pentru anumiți vorbitori rezultatele sunt acceptabile în timp ce eroarea pentru alți vorbitori este foarte mare, acest lucru ducând la o traducere greșită. Această rata de eroare foarte mare este dată de caracteristicile diferite ale vorbitorilor (timbru, pronunție, dicție, etc.)

Observație : Sistemul de recunoaștere a fost antrenat cu 8 densități Gaussiene corespunzătoare a 100 de senone.

100 senone 8 densitati	
SPEAKER ID	WER [%]
227	86.1
251	65.5
269	21.4
274	14.6
276	32.8

Tabelul 3. 4 Evaluarea sistemului antrenat cu date provenite de la un singur vorbitor și testat cu date de intrare provenite de la 5 vorbitori diferiți.

Tabelul 3.5 prezintă informațiile obținute în urma antrenării sistemului cu date provenite de la 50 de vorbitori diferiți și testat cu date de intrare provenite de la 17 vorbitori.

Nr. Senone	100	200
Gaussians nr.	WER[%]	
1	19.3	39
2	12.6	15.6
4	7.8	11.3
8	6.3	12
16	5.7	
32	5.5	
64	5.7	

Tabelul 3. 5 Evaluarea sistemului antrenat cu date provenite de la 50 de vorbitori și testat cu date de intrare provenite de la 17 vorbitori diferiți.

Se poate observa că odată cu creșterea numărului de densități Gaussiene, rata de eroare a cuvintelor este foarte mică relativ la datele din tabelul 3.4 .

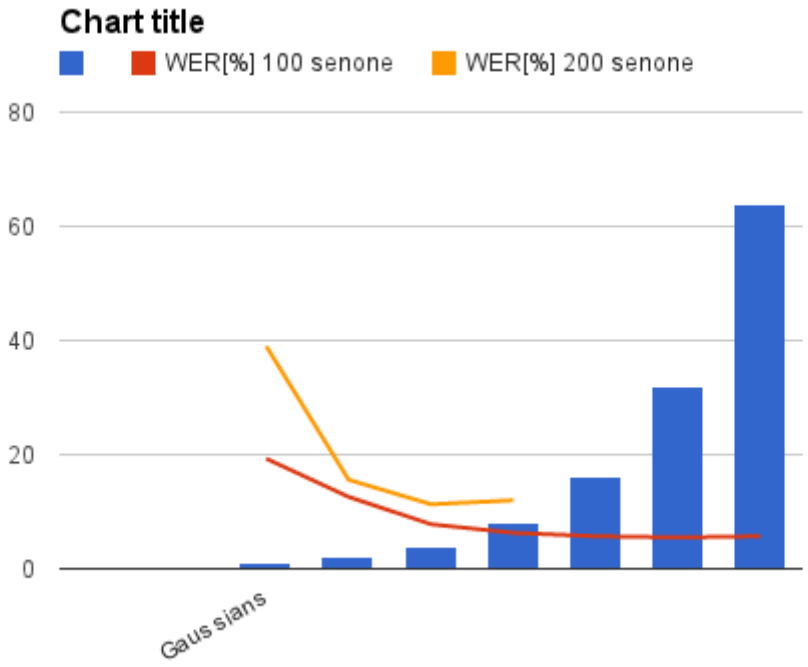

Fig 3. 7 Evaluarea sistemului de recunoaștere cu antrenare pentru 50 de vorbitori și testare pt 17 vorbitori .

3.3 Dezvoltarea unui sistem de recunoaștere a vorbirii continue provenite de la diferite posturi de radio cu ajutorul sistemului CMU Sphinx.

În dezvoltarea sistemului de recunoaștere a vorbirii continue cu date de intrare de la diferite posturi radio, s-a urmărit, în primul rând, achiziția de semnal audio prin intermediul stream-urilor audio puse la dispoziție de posturile de radio (emise prin intermediul internetului). Construirea modelelor acustice, fonetic și a gramaticii sistemului nu au făcut obiectul proiectului curent. Aceste resurse au fost puse la dispoziție de către profesorul îndrumător. În vederea înțelegerii modului de funcționare al sistemului de recunoaștere, s-a impus dezvoltarea unui sistem de tip demo cu sarcina de recunoaștere a cifrelor (Cap. 3.2) , necesar pe viitor optimizării sistemului de recunoaștere continuu dezvoltat.

3.3.1 Implementarea unei componente de achiziție a semnalului audio provenit de la diferite posturi de radio

În figura 2.1 este prezentată arhitectura generală a unui sistem de recunoaștere CMU Sphinx. Se poate observa că prima componentă estențială este FrontEnd, componenta ce se ocupă de achiziția semnalului și extragerea parametrilor cepstrali necesari decodării. Frontend, așa cum este descrisă în manualul CMU Sphinx, este o clasă ce împachetează diferite componente pentru lanțul de procesare. Front-endul este modelat ca o serie de procesoare, fiecare îndeplinind o funcție specifică de prelucrare a semnalului. De exemplu, un procespr execută Fast Fourier Transform (FFT) pe date de intrare, un alt procesor efectuează filtrarea trece-sus, etc. Figura 3.8 descrie modul de procesare al front-endului :

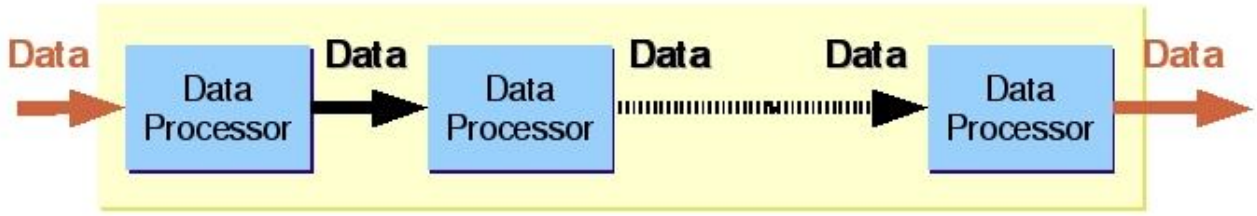

Fig 3. 8 CMU Sphinx4 Front-end

Fiecare componentă de tip Data Processor descrisă în figură implementează interfața DataProcessor. Obiectele care implementează interfața Data sunt date de intrare și ieșire ale front-endului, acestea trecând prin fiecare procesor configurat. Datele de intrare sunt de obicei semnale audio. Similar, datele de ieșire sunt, în general, o serie de parametri extrași din datele de intrare, dar acest mod este ușor configurabil în Sphinx4.

Componenta front-end folosește un model de tip pull. Prin apelarea metodei getData asupra acestei componente, se acționează această metodă asupra penultimului procesor (DataProcessor) și, în mod recursiv, această metodă este apelată până la primul procesor de date. Intrarea front-end-ului este, de obicei, un alt procesor de date.

Așadar, s-a impus dezvoltarea unei clase de tip DataProcessor ce va avea ca date de intrare stream-ul audio provenit de la posturile de radio și ca date de ieșire obiecte de tip Data ce vor parcurge în continuare lanțul front-end-ului.

Pentru efectuarea acestei sarcini a fost configurata o clasă, componenta LiveStreamDataSource, ce poate accepta ca intrare un stream audio asupra căruia vom apela metoda getData() descrisă mai sus. Această metodă va converti stream-ul de intrare în obiecte de tip Data ce vor fi, în continuare, procesate de celelalte componente front-end.

Pentru sarcina de procesare, formatul stream-ului audio va trebui să fie :

- rata de eșantionare 16 kHz

- numărul de bit/sample 16

- semnalul va trebui să fie pe un singur canal (mono)

Figura 3.9 ilustrează metoda setLiveStreamURL, metodă ce primește ca parametri de intrare numele și adresa unui stream audio, analizează datele de intrare și, după caz, convertește acest semnal pentru forma dorită.
Observație : Constantele SAMPLE_RATE, BITS_PER_SAMPLE, NO_CHANNELS, au fost definite în prealabil, luând valorile prezentate mai sus

```
99  /**
100  * Sets the InputStream from which this LiveStreamDataSource reads.
101  *
     * @param inputStream the InputStream from which audio data comes
     * @param streamName the name of the InputStream
104  */
     public void setLiveStreamURL(String _streamName, String _streamURL) {
106      streamURL = _streamURL;
107      streamName = _streamName;
108      streamEndReached = false;
109      utteranceEndSent = false;
110      utteranceStarted = false;
111      totalValuesRead = 0;
112
113      //Getting the audioInputStream from the URL
114      AudioInputStream _audioInputStream = pullAudioStream(_streamURL);
115
116      //Converting the audio input stream into a stream with the fixed params: AUDIO_ENCODING, NO_CHANNELS, SAMPLE_RATE, BIG_ENDIAN
117      // System.out.println("Initial format " + _audioInputStream.getFormat());
118      if (_audioInputStream.getFormat().getEncoding() != AUDIO_ENCODING) {
119          logger.log(Level.INFO, "Changing encoding from {0} to {1}", new Object[]{_audioInputStream.getFormat().getEncoding(), AUDIO_ENCODIN
120          _audioInputStream = convertStreamEncoding(AUDIO_ENCODING, _audioInputStream);
121      }
122      if (_audioInputStream.getFormat().getChannels() != NO_CHANNELS
123              || _audioInputStream.getFormat().getSampleRate() != SAMPLE_RATE
124              || _audioInputStream.getFormat().isBigEndian() != BIG_ENDIAN) {
          logger.info("Changing number of channels from " + _audioInputStream.getFormat().getChannels() + " to " + NO_CHANNELS);
          logger.info("Changing sample rate from " + _audioInputStream.getFormat().getSampleRate() + " to " + SAMPLE_RATE);
          logger.info("Changing big-endian from " + _audioInputStream.getFormat().isBigEndian() + " to " + BIG_ENDIAN);
128       _audioInputStream = convertStreamFormat(SAMPLE_RATE, NO_CHANNELS, BIG_ENDIAN, _audioInputStream);
129   }
130
131   dataStream = _audioInputStream;
132 }
```

Fig 3. 9 Metoda setLiveStreamURL

Așa cum am precizat, componentele din front-end CMU Sphinx se bazează pe modelul pull. Se observă că semnalul audio este 'tras' cu ajutorul căii specificate și apoi analizat. Această metodă a fost de asemenea implementată anterior și este descrisă în figura 3.10

```
134  public AudioInputStream pullAudioStream(String _stringStreamURL) {
135      try {
136          URL _streamURL = new URL(_stringStreamURL);
137          Socket _socket = new Socket(_streamURL.getHost(), _streamURL.getPort());
138          OutputStream _outputStream = _socket.getOutputStream();
139          String _userAgent = "WinampMPEG/5.1";
140          String _request = "GET / HTTP/1.0\r\nuser-agent: " + _userAgent
141                  + "\r\nAccept: */*\r\nConnection: keep-alive\r\n\r\n";
142          _outputStream.write(_request.getBytes());
143
144          //Creating the live stream input stream
145          /* without additional libraries in classpath (jaad-0.8.4.jar, jl1.0.1.jar, mp3spi1.9.5.jar and tritonus_share.jar)
146           * the following inputStream to audioInputStream conversion does not work for MPEG streamss!
147           */
148          InputStream _inputStream = new BufferedInputStream(_socket.getInputStream());
149          AudioInputStream _audioInputStream = AudioSystem.getAudioInputStream(_inputStream);
150          // System.out.println("Stream OK!");
151          return _audioInputStream;
152      } catch (Exception _exc) {
153          _exc.printStackTrace();
154      }
155      return null;
156  }
```

Fig 3. 10 Metoda *pullAudioStream* folosită pentru pentru extragerea semnalului de intrare .

Se observă că pentru un nou stream audio, se deschide un nou socket java prin care se recepționează datele emise de către postul radio. Această metodă returnează stream-ul audio sub forma *AudioInputStream.* Aceasta este o clasă implicită Java ce tratează stream-urile audio.

Deoarece această clasă creată implementează interfața BaseDataProcessor din CMU Sphinx4, interfață ce implementează metoda getData() descrisă anterior, putem apela această metodă pentru stream-ul radio introdus urmând ca la o cerere de tip pull asupra acestei componente din front-end, clasa să returneze obiecte de tip Data din stream-ul audio dorit.

Având în vedere faptul că, pentru evaluarea manuală a transcrierii și eventuale corecturi, furnizorii de informații radio nu oferă posibilitatea consultării unei arhive, s-a impus dezvoltarea unei alte componente front-end ce poate scrie stream-ul audio pe disc.

Această componentă a fost denumită *EnhacedWavWriter*. Fiind definită ca o componentă front-end, s-a impus ca datele de intrare să fie de tip Data, iar transcrierea scrisă pe disc să aibă formatul *.wav* . Componenta ce scrie datele pe disc va avea ca parametri de intrare calea către fișierul de scriere și obiectele de tip Data ce vor constitui fișierul audio. Pentru obținerea obiectelor, vom efectua o cerere de tip pull către componenta anterioara din lanțul Front-end .

Obiectele de tip Data vor trebui reconvertite în stream audio ce va fi scris pe disc. Pentru a efectua această cerință, vom apela din nou la clasa AudioInputStream și vom extrage din matricea de biți a obiectului Data recepționat la intrare. Așadar, se va defini numele fișierului audio ce va conține data acutală a scrierii pe disc și calea către fișier și se va constitui fișierul .wav ca *AudioInputStream*.

Pentru scrierea fișierului de tip .wav, s-a impus ca rata de eșantionare să rămână 16 kHz, iar dimensiunea unui eșantion să fie 16 bit/eșantion. Alte opțiuni de configurare a acestei clase, în afara căii de acces, vor fi dimensiunea fișierului (în acest caz s-a recurs la crearea unui fișier cu

dimensiunea de 30 de secunde) și eventuale modificări asupra ratei deeșantionare pentru ca fișierul audio să ocupe un spațiu mai mic pe disc.

Toate informațiile referitoare la această componentă se gasesc în pachetul org.etti.sphinx.frontend.util (codul atașat pe cd).

Aceste componente descrise vor fi apoi introduse în lanțul normal de prelucrare CMU Sphinx prin configurarea front-end-ului .

Observație: Deoarece următoarele componente din lanțul CMU Sphinx vor necesita date de intrare de tip Data, aceste componente sunt dezvoltate astfel încât să returneze în continuare acest tip de date.

3.3.2 Arhitectura serverului Java

Pentru comunicația cu potențialii clienți ce vor solicita transcrierea diferitelor posturi de radio au fost implementate două metode. Prima metodă este reprezentată de comunicația cu ajutorul Java socket. Deoarece această metodă nu beneficiază decât de dezvoltarea unui client de tip dummy, necesar testării implementării, această metodă nu va fi decât amintită în acest document. Mai multe informații se pot obține consultând codul pachetului org.etti.speech.liveradioasr.server.

Metoda de comunicare preferată a fost implementată cu ajutorul protocolului http, deoarece scalabilitatea acestui protocol în aplicațiile web este mult mai mare, iar numărul aplicațiilor ce pot beneficia de acest serviciu este superior. Totuși, metoda de comunicare cu ajutorul Java socket poate folosi pentru comunicarea cu diverse mașini, microcontrollere, etc .

Așa cum am menționat în capitolul introductiv, serviciul poate fi folosit pentru orice post de radio ce emite în limba română. Este de preferat ca subiectul general abordat să fie discursul cursiv și nu prezentarea de muzică, deoarece sarcina acestui sistem este de a transcrie vorbirea continuă.

Pentru configurarea radiourilor s-a recurs la crearea unei liste de radiouri cu diferite informații specifice și necesare sistemului dintre care amintim : calea de accesare a stream-ului audio (URL), un nume scurt asociat identificării și un template pentru fișierul destinație al transcrierii. Lista se poate găsi în fișierul config/radiostations.list și poate fi modificată cu orice editor de text în vederea obținerii altor transcrieri. Pentru demo-ul aferent acestui proiect, am ales o listă de 4 radiouri .

Deoarece comunicarea prin intermediului socket presupune codificarea informațiilor de tip obiect Java în cod binar, iar comunicarea cu ajutorul protocolului HTTP presupune formatarea datelor sub un format ales (text, JSON, etc.) au fost create anumite clase speciale ce pot facilita primul tip de comunicare prezentat. Aceste clase dispun de metode ce returnează doar textul transcrierii. Vor face obiectul prezentării doar aceste metode deoarece prin alegerea protocolului HTTP, doar informațiile sub formă de text au fost necesare.

În continuare va fi prezentată clasa RadioTranscriberServer din pachetul org.etti.speech.liveradioasr.server, clasă al cărui obiectiv este derularea procesului de transcriere pentru posturile de radio configurate.

Deoarece există opțiunea de configurare a diferitelor posturi radio și a comunicării prin intermediul Java sockets, s-a impus dezvoltarea unei clase primare ce va derula procesul de transcriere și, în cazul comunicării Java sockets va stabili comunicarea unui client nou prin deschiderea unui nou socket. De asemenea, această clasă va gestiona și crearea de fișiere structurate unde vor fi salvate transcrierile și fișierele audio aferente. O altă carateristică foarte importantă a acestei clase este adăugarea de notificări.

Figura 3.11 descrie metoda de prelucrare a listei de radiouri și formarea unei liste cu informații specifice. Fiecare post de radio va fi un obiect de tip *RadioStationInfo* . Clasa *RadioStationInfo* descrie un post de radio, cu toate informațiile puse la dispoziție în configurare. Această clasă implementează metode de tip *get* și *set* pentru aceste proprietăți pentru atribuirea de valori și accesarea acestora.

Deoarece transcrierile vor fi stocate pe disc, la denumirea fișierului vom adăuga o serie de date utile identificării ulterioare (id-ul specific radioului și data creării). Acest lucru este realizat cu o serie de metode din clasa *RadioStationInfo* . Mai multe informații pot fi accesate în pachetul org.etti.speech.liveradioasr.struct

```
67      private void parseRadioStationsListFile(String _radioStationsInfoFile)
68              throws IOException, LineUnavailableException {
69
70          info("Parsing file " + _radioStationsInfoFile + ' ');
71
72          Properties properties = new Properties();
73          properties.load(new FileInputStream(_radioStationsInfoFile));
74
75          String radioStationsLine = properties.getProperty("radioStations");
76          String[] radioStationsName = radioStationsLine.split("\\s+");
77
78          for (String name : radioStationsName) {
79              // Radio Fullname
80              String radioName = properties.getProperty(name + ".fullName");
81              if (radioName == null) {
82                  throw new NullPointerException("No Name for Radio " + name);
83              }
84              // Radio URL
85              String URL = properties.getProperty(name + ".URL");
86              if (URL == null) {
87                  throw new NullPointerException("No URL for Radio  " + name);
88              }
89              // Logo Path
90              String logoPath = properties.getProperty(name + ".logoPath");
91              if (logoPath == null) {
92                  throw new NullPointerException("No Logo path for Radio  " + name);
93              }
94              // Short Name
95              String shortName = properties.getProperty(name + ".shortName");
96              if (shortName == null) {
97                  throw new NullPointerException("No Short name for Radio  " + name);
98              }
99              // Transcription File Name Template
00              String transFileNameTemplate = properties.getProperty(name + ".transcriptionFileNameTemplate");
01              if (transFileNameTemplate == null) {
02                  throw new NullPointerException("No Transcription Template name for Radio  " + name);
03              }
04              // add the fullname , URL , logoPath & shortName of the radio to the radioStationsList
05              radioStationsList.add(new RadioStationInfo(radioName, URL, logoPath, shortName, transFileNameTemplate, transcriptionsFolderN
06          }
```

Fig 3. 11 Prelucrarea listei de radiouri și crearea de obiecte specifice fiecărui radio

Având la dispoziție toate datele de identificare specifice radiourilor, procesul de decodare poate începe. La rularea clasei principale (*RadioTranscriberServer*), pentru fiecare post de radio se va deschide un thread nou ce va constitui procesul de decodare al semnalului audio.

Aceste thread sunt descrise într-o clasă separată denumită generic *RadioTranscriberThread* (pachetul org.etti.speech.liveradioasr.struct).

Reamintim că odată cu procesarea fișierului de configurare al radiourilor, a fost creată o listă de obiecte de tip *RadioStationInfo*. Prin parcurgerea acestei liste se poate iniția un proces nou corespunzător fiecărei intrări.

La inițializarea unui proces nou de decodare, specific unei intrări în lista de radiouri, se va instanția sistemul de recunoaștere CMU Sphinx și acesta va începe decodarea semnalului audio recepționat.

Un alt element de structură foarte important este implementarea unui centralizator al transcrierilor ce are rolul de a verifica apariția de conținut nou, scrierea acestuia pe disc și apelarea unei metode din server pentru adăugarea de transcrieri în lista clienților.
Figura 3.12 ilustrează structura și circulația datelor în acest sistem

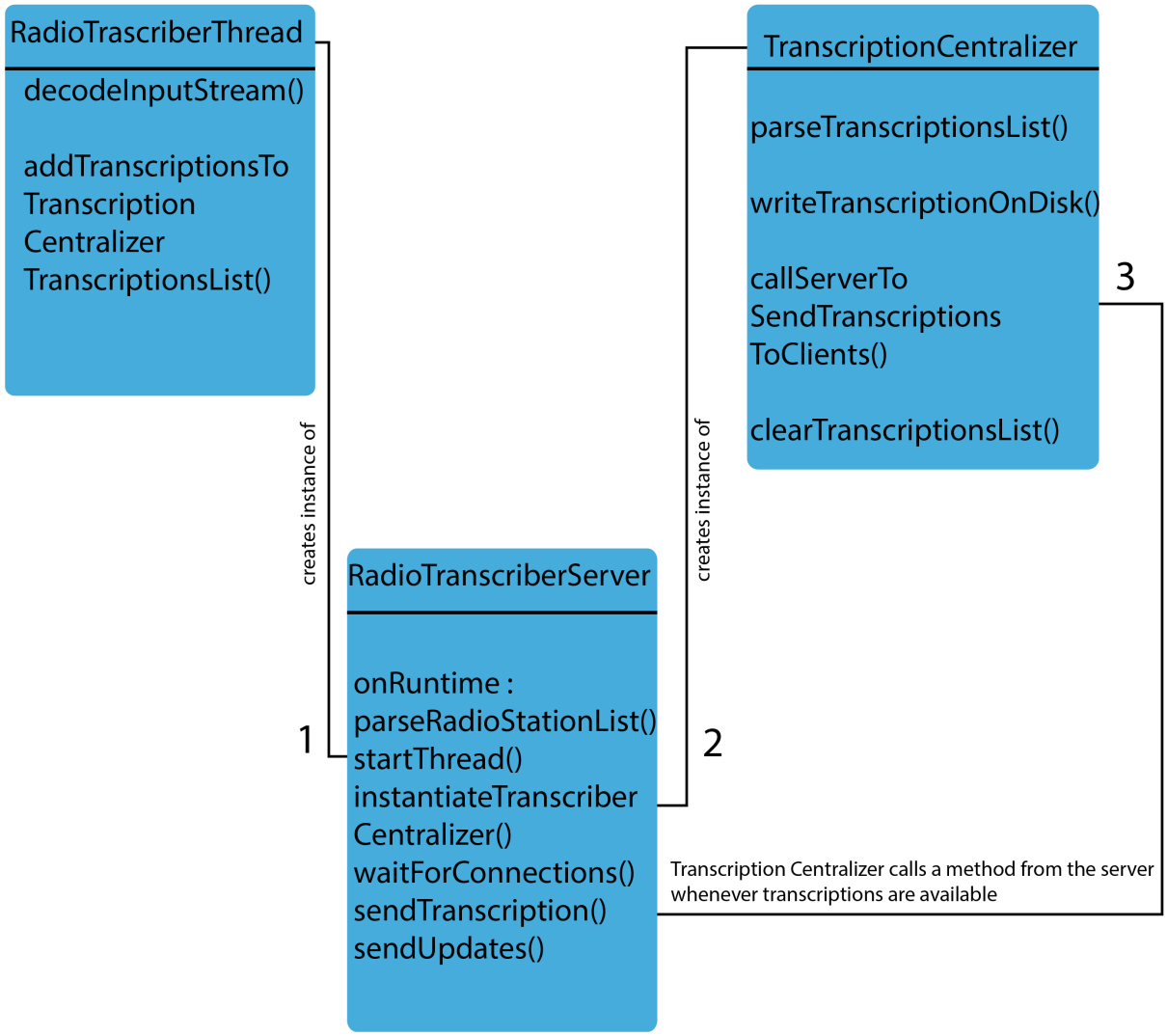

Fig 3. 12 Procesele executate odată cu rularea clasei *RadioTranscriberServer*

Așadar, la rularea clasei *RadioTranscriberServer* se va parcurge și popula lista de radiouri, se vor instanția atâtea clase de tip *RadioTranscriberThread* câte intrări vor fi în lista de radiouri și se va inițializa un obiect de tip *TranscriptionCentralizer*.

Clasa *RadioTranscriberThread* ce extinde clasa Java Thread va iniția sistemul CMU Sphinx prin configurarea front-end-ului cu informațiile necesare achiziției de semnal de la un radio și va iniția decodarea acestui semnal. La apariția de rezultate noi, va popula lista de transcrieri din *TranscriptionCentralizer* cu obiecte de tip *TranscriptionUpdate* , un alt element structural necesar comunicării între aceste clase. La rândul său, centralizatorul aștepta o perioadă de timp pentru popularea listei de transcrieri, după care va parcurge această listă, va scrie pe disc transcrierile recepționate și va apela metoda *sendTranscriptionToHttpClients()* din *RadioTranscriberServer* . Această metodă va verifica dacă sunt disponibile transcrieri pentru clienții conectați și, în cazul în care aceste update-uri sunt disponibile, le va pune la dispoziție în vederea trimiterii.

După ce se încheie parcurgerea listei de transcrieri în centralizator, această listă se va goli evitându-se astfel scrierea pe disc a aceleiași transcrieri și trimiterea către client.

Figura 3.13 arată modul concret în care se efectuează procesul de decodare și populare a listei de transcrieri, iar figura 3.14 prezintă modul de parcurgere al listei, procesarea informațiilor și golirea acesteia.

```
37      public RadioTranscriberThread(RadioTranscriberServer _server, RadioStationInfo _info, String _configFileName,
38              String _speechDataFolder) throws Exception {
39
40          parent = _server;
41          radiosInfo = _info;
42
43          threadTranscriptionCentralizer = parent.getTranscriptionCentralizer();
44          URL _configURL = new File(_configFileName).toURI().toURL();
45          configManager = new ConfigurationManager(_configURL);
46          continuousSpeechRecognizer = ((Recognizer) configManager.lookup("freeSpeechRecognizer"));
47          continuousSpeechRecognizer.allocate();
48          enhacedWavWritter = ((EnhancedWavWriter) configManager.lookup("wavWriter"));
49          enhacedWavWritter.setDatedOutFilePattern(true);
50          enhacedWavWritter.setIsCompletePath(true);
51          enhacedWavWritter.setOutFilePattern(radiosInfo.getRadioShortName());
52          enhacedWavWritter.setOutFolder(parent.getAudioFoldername());
53          String _logFileName = setupLogFileName();
54          setupLogger(_logFileName);
55      }
56  //------------------------------------------------------------------------------
57
58      @Override
59      public void run() {
60          // configure the audio input for the recognizer
61          LiveStreamDataSource _liveDataSource = (LiveStreamDataSource) configManager.lookup("liveStreamDataSource");
62          _liveDataSource.setLiveStreamURL(radiosInfo.getRadioName(), radiosInfo.getRadioURL());
63          Result result;
64          while ((result = continuousSpeechRecognizer.recognize()) != null) {
65              String resultText = result.getBestResultNoFiller();
66              //System.out.println(streamName + ":  " + resultText);
67              threadTranscriptionCentralizer.addTranscriptionUpdate(new TranscriptionUpdate(radiosInfo.getRadioShortName(), resultText
68          }
69      }
70  //------------------------------------------------------------------------------
```

Fig 3. 13 Constructorul clasei *RadioTranscriberThread* și rularea procesului de decodare.

Se observă că odată cu instanțierea unui obiect de tip *RadioTranscriberThread* se face legătura cu centralizatorul transcrierilor și se instanțiază obiectele necesare configurării sistemului de decodare.

Odată cu rularea acestei metode, se configurează Front-end-ul cu informațiile necesare achiziției de semnal și începe procesul de decodare. Odată cu recepționarea rezultatelor de la decodor, acestea sunt adăugate în lista de transcrieri a centralizatorului sub forma unor obiecte de

tip *TranscriptionUpdate* (linia de cod 67). Mai multe informații despre acest tip de obiecte pot fi accesate consultând codul din pachetul org.etti.speech.liveradioasr.struct, clasa *TranscriptionUpdate* .

```
28      public TranscriptionCentralizer(RadioTranscriberServer _server) {
29          parent = _server ;
30          transcriptionUpdates = new ArrayList<>();
31      }
32    //---------------------------------------------------------------------------
33      @Override
35      public void run() {
36          while (true) {
38              try {
                    Thread.sleep(15000);
                    synchronized(this){
                        for (TranscriptionUpdate _transcript : transcriptionUpdates) {
40                          System.out.println(_transcript.getRadioShortName() + "  " + _transcript.getUpdatedTranscription()
41                          writeTranscriptionsOnDisk(_transcript);
42                          //parent.sendTranscriptionUpdateToClientPeers(_transcript);
43                          parent.sendTranscriptionUpdateToHttpClients(_transcript);
44                      }
45                      transcriptionUpdates.clear();
46                  }
47              } catch (IOException | InterruptedException ex) {
                    ex.printStackTrace();
49                  Logger.getLogger(TranscriptionCentralizer.class.getName()).log(Level.SEVERE, null, ex);
50              }
51          }
52      }
53    //---------------------------------------------------------------------------
```

Fig 3. 14 Contructorul clasei *TranscriptionCentralizer* și modul de funcționare al acesteia

Așa cum a fost descris anterior, centralizatorul așteaptă popularea listei de transcrieri, după care recurge la parcurgerea acesteia, scrierea transcrierii pe disc și apelarea metodei de trimitere a transcrierii către clienți.

Metoda *sendTranscriptionUpdateToHttpClients* din server nu va trimite transcrierea direct către clienții conectați. De acest lucru se ocupă server-ul HTTP Glasfish ce va fi descris în secțiunea următoare. Această metodă iterează lista de clienți conectați, verifică dacă transcrierea apărută este cea solicitată de fiecare din clienți și, în caz pozitiv, adaugă transcrierea la o listă specifică fiecărui client. Această procedură este descrisă în detaliu în secțiunea 3.3.3 (Arhitectura server-ului HTTP Glasfish).

3.3.3 Arhitectura server-ului HTTP Glassfish din proiectul Live Radio ASR

Implementarea comunicării cu ajutorul protocolului HTTP a impus dezvoltarea unui server ce poate servi cereri de tip GET solicitate de clienți HTTP. Acest lucru a fost implementat cu ajutorul pachetului Glassfish.

De asemenea s-a impus generarea pachetului .jar a proiectului prezentat mai sus și adăugarea acestuia în lista de dependențe a server-ului HTTP (pentru accesarea claselor și funcționalității). Odată cu adăugarea acestui proiect, a fost necesară adăugarea tuturor dependențelor proiectului de recunoaștere Java .

Odată configurată lista de dependențe, a fost posibilă implementarea acestui server, cu posibilitatea răspunderii la cereri de tip GET prin metoda server-sent events. Această metodă a fost descrisă pe larg în capitolul 2, secțiunea 2.15.

La pornirea server-ului HTTP, se va instanția și rula clasa *RadioTranscriberServer* a cărei funcționalitate a fost, de asemenea, descrisă anterior și se va începe procesul de decodare, așteptându-se conectarea clienților.

Implementarea sistemului SSE presupune descrierea și înregistrarea unei clase resursă. Această clasă trebuie să implementeze procesarea cererilor și modul de răspuns la acestea. Concret, a fost dezvoltată o clasă ce va răspunde cererilor HTTP de tip GET, va procesa acest tip de cerere (va extrage radioul dorit de către client) și va furniza transcrierea disponibilă din ziua curentă urmând să funizeze diferite update-uri odată cu apariția acestora. Clientul va fi înregistrat într-o listă de clienți de tip *RadioTranscriberClient* . Această clasă este generică pentru descrierea unui client, iar câmpurile sunt populate cu informații generale: radio-ul pentru care a fost cerută transcrierea și informații de identificare a clientului în mediul web (adresa fizică, IP).

După trimiterea transcrierii disponibile pentru ziua respectivă, procesul va aștepta un timp configurabil (1000 ms în cazul nostru) după care va verifica popularea listei de update-uri specifice clientului și stocată în instanța acestuia (această listă este populată de către *RadioTranscriberServer* iar procesul de populare a fost descris mai sus). În cazul în care există intrări în această listă, ele vor fi procesate și împachetate ca un nou eveniment ce va fi trimis clientului, după transmitere lista fiind depopulată.

Pentru trimiterea inițială a transcrierii a fost implementată o metodă specifică ce accesează fișierul de stocare al transcrierii, îl parcurge și îl returnează sub formă de obiect *TranscriptionResponse* . Acest tip de obiect permite returnarea conținutului sub formă de text.

Figurile 3.15 și 3.16 ilustrează modul de parcurgere a fișierului și trimiterea acestuia către client.

```
103   //-------------------------------------------------------------------------------------
104       /**
105        * This method sends TranscriptionResponse objects to client consisting in a
106        * list of transcriptions for the whole day
107        *
108        * @param _radioStationShortName ...
109        * @throws FileNotFoundException
110        * @throws IOException
111        */
          private TranscriptionResponse sendRadioTranscriptionResponse(String _radioStationShortName) throws FileNotFoundException, IOException
113           //Obtain the transcription file name
114           parent = App.currentRunningServer;
115           String _fileName = null;
116           for (RadioStationInfo _radioInfo : parent.getRadioStationsInfo()) {
117               if (_radioStationShortName.equals(_radioInfo.getRadioShortName())) {
118                   _fileName = _radioInfo.getTranscriptionCompleteFilePath(new Date());
119               }
120           }
121
122           //Read the transcription file
              TranscriptionResponse _response = new TranscriptionResponse(Files.readAllLines(Paths.get(_fileName), Charset.defaultCharset()));
124           return _response;
125       }
126   }
```

Fig 3. 15 Parcurgerea fișierului de transcriere și returnarea rezultatului sub forma unui obiect *TranscriptionResponse* .

Această metodă admite ca parametru de intrare o proprietate a unui radio, va itera lista de radiouri disponibilă în server și va extrage calea către fișierul ce conține transcrierea, apoi va parcurge fișierul linie cu linie și va returna obiectul ce conține transcrierea.

```
                for (String line : sendRadioTranscriptionResponse(radioName).getTranscription()) {
61                  final OutboundEvent.Builder eventBuilder
62                  = new OutboundEvent.Builder();
63                  eventBuilder.data(String.class,
64                      line);
65                  final OutboundEvent event = eventBuilder.build();
66                  eventOutput.write(event);
67                  }
```

Fig 3. 16 Trimiterea transcrierii disponibile pentru ziua curentă către client sub unei succesiuni de evenimente.

După această operațiune, server-ul așteaptă un timp, și apoi încearcă trimiterea update-urilor către client după cum urmează :

```
68              while (true) {
69                  // waits for toBeSentTranscriptionUpdates to be populated
                    Thread.sleep(1000);
71                  if (!(_radioClient.getToBeSentTranscriptionUpdates().isEmpty())) {
72                      List<TranscriptionUpdate> toBeSentTranscriptionUpdates = _radioClient.getToBeSentTranscriptionUpdate
73                      for (TranscriptionUpdate _transcUpdate : toBeSentTranscriptionUpdates) {
74                          //outputStream.writeObject(_transcUpdate);
75                          OutboundEvent.Builder eventBuilder
76                                  = new OutboundEvent.Builder();
77                          //eventBuilder.name("message-to-client");
78                          eventBuilder.data(String.class,
79                                  _transcUpdate.getUpdatedTranscription());
80                            final OutboundEvent event = eventBuilder.build();
81                            eventOutput.write(event);
82                      }
83                      toBeSentTranscriptionUpdates.removeAll(toBeSentTranscriptionUpdates);
84                  }
85                  }
```

Fig 3. 17 Transmiterea update-urilor către client sub forma unor noi evenimente.

3.3.4 Clientul Web

Dezvoltarea clientului web a pornit de la premisa realizării unui sistem de transcriere în timp real. Această sarcină este, în continuare, greu de atins cu resursele actuale. Principalele caracteristici ale clientului sunt : ascultarea postului de radio dorit și vizualizarea transcrierii. Pentru asta, va trebui interpretat evenimentul trimis de către server și afișat într-o zonă text. Clientul implementat pentru demo este, de asemenea, un exercițiu de design web executat în Adobe Photoshop și apoi implementat cu ajutorul html5 și css3 . Figura următoare prezintă codul propriu-zis pentru tratarea evenimentelor de tip eventSource. În cazul în care aceste evenimente nu sunt suportate de browser sau la întâmpinarea unei erori de conectare la server-ul Java, clientul va primi o alertă cu mesajul

specific. Odată cu apariția mesajului, textul va fi adăugat la căsuța de text.

```
if(eventSource !== undefined){
    eventSource.close();
}
    if((typeof EventSource)==="undefined")
            {
                alert("display","SSE Not supported on your browser");
                return;
            }
eventSource = new EventSource($eventSourceURL);
eventSource.onopen = function (e) {
    console.log("Waiting message..");
};
eventSource.onerror = function (e) {
    alert("Transcription for selected radio is not available");
};

eventSource.onmessage=function (e) {
    addtext(" ");
    addtext(e.data);
};

}
```

Fig 3. 18 Receptarea și afișarea transcrierii la clientul web

Pentru partea de redare audio, am folosit librăria SoundManager, iar diferite alte funcționalități (oprirea radioului, simularea unui equalizer, etc) au fost implementate cu funcții Javascript ce pot fi găsite în fișierul Demo.html și assets/js/equalizer.js .

Contribuții personale

Contribuțiile personale au constat în dezvoltarea sistemului de recunoaștere a vorbirii continue, cu ajutorul modelului acustic și de limbă puse la dispoziție de către profesorul coordonator. Pentru implementarea acestei componente am folosit librăria CMU Sphinx4 descrisă, de asemenea, în documentul curent.

Următorul pas a fost implementarea unei arhitecturi care să respecte modelul client-server și modelul RESTful pentru server. În capitolul 3 sunt explicate elementele de structură și implementare. Toate aceste elemente au fost dezvoltate personal.

Design-ul și implementarea în tehnologii web a interfeței grafice a fost de asemenea o sarcină importantă. Dezvoltarea a pornit de la elemente de design în soft-ul Adobe Photoshop, până la implementarea finală (HTML, CSS). Alături de interfața grafică, am dezvoltat un modul de autentificare capabil să suporte mediul social. Dezvoltarea interfeței grafice a presupus dezvoltarea de funcții Javascript și JQuery pentru a putea oferi utilizatorului posibilitatea să asculte posturile de radio și să consulte în același timp transcrierea. Pe lângă aceste funcționalități de bază, au fost adăugate diferite butoane pentru a opri postul de radio sau pentru a ajusta volumul. Pentru acestea am dezvoltat o serie de funcții ce sunt accesibile consultând codul de pe cd.

Un ultim pas în implementare a fost configurarea unui server Linux, pus la dispoziție de profesorul coordonator , pentru găzduirea și publicarea proiectului final. Accesând proiectul, veți găsi diferite informații despre implementare, o legătură foarte utilă cu o rețea de partajare a proiectelor denumită GitHub și accesul la demo.

Capitolul 3 prezintă în detaliu modul de implementare și arhitectura generală a proiectului.

Concluzii și direcții de îmbunătățire

Proiectul a reușit să îndeplinească sarcina de transcriere a vorbirii continue provenită de la diferite surse de intrare radio. Datorită duratei de achiziție și procesare a semnalului, s-a înregistrat o întârziere de aproximativ 10 secunde față de transcrierea în timp real. Pentru îmbunătățirea acestor valori, vor fi prezentate câteva direcții privind implementarea unui tool de diarizare numit Lium ca și componentă de front-end pentru recognizer. Momentan, această unealtă este implementată, dar prin modificarea anumitor funcții de filtrare și studierea cu atenție a modului de acțiune a acestui pachet, cred că rezultatele pot fi net îmbunătățite.

Componenta descrisă mai sus se gasășește în pachetul org.etti.speech.frontend.util și se numește *SpeakerDiarizationSpeechMarker* . Rolul principal al acestei componente front-end este de a filtra semnalul de la intrare înainte de a fi redirecționat către motorul de decodare. Această filtrare presupune eliminarea zgomotelor, a momentelor de liniște sau a celor ce conțin muzică sau sunete ce nu fac obiectul recunoașterii. Funcția de filtrare cu care a fost implementată componenta se află în același pachet și se numeste *SFilter2* .Odată ce semnalul este cât mai curat înainte de a fi decodat, rezultatele se vor îmbunătăți.

O altă direcție este testarea acestui protocol de comunicare cu alte limbaje de programare ce implementează modulul HTTP. Momentan, testarea a fost efectuată doar prin implementarea evenimentelor în limbajul Javascript. Arhitectura proiectului permite și comunicarea pe socket . Este menționat că aceast tip de comunicare nu a fost testat decât cu un client minimal, dar acest lucru poate fi extins. Să presupunem cazul înregistrării transcrierilor în depozite de date (data-warehouse). În acest caz se preferă comunicarea cu ajutorul Java sockets, dezavantajul constând în faptul că mașina ce va primi transcrierea va trebui să știe să decodeze obiectele de tip Java trimise de server-ul de transcriere.

Momentan, textul nu este formatat corespunzător respectând standardele limbii române. Sistemul trimite către client textul neprocesat.

Putem considera o altă direcție de îmbunătățire trimiterea stream-ului audio către client direct de pe server-ul de transcriere, etichetat cu ștampile de timp, astfel încât, odată cu recepționarea textului, clientul să poată apăsa pe unul din cuvinte și să poată reasculta fragmetul audio (implicit, textul va fi etichetat cu ștampile audio).

Aceste direcții pot fi implementate și cu ajutorul server-ului NodeJS (transcrierea poate fi formatată de către această componenta, în vederea distribuirii sarcinilor).

Privind interfața grafică accesibilă utilizatorului, acesta poate avea în viitor o secțiune dedicată unde poate salva anumite transcrieri, le poate edita, etc.

Aplicația poate fi modificată astfel încât utilizatorul să poată introduce un URL în interfața grafică, iar server-ul să înceapă un nou thread pentru a putea procesa această cerere (în cazul în care un utilizator dorește să aibă acces la o emisiune radio, dar timpul nu îi permite).

Bibliografie :

[1],[9] Research and Development Project in Spoken Language Technology – Horia Cucu

[2] http://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-345-automatic-speech-recognition-spring-2003/lecture-notes/

[3] http://www.voxforge.org/home/docs/faq/faq/what-is-an-acoustic-model

[4] http://cmusphinx.sourceforge.net/wiki/tutorialam

[5] [Bahl, 1991] Bahl, L.R., et al., "Multonic Markov Word Models for Large Vocabulary Continuous Speech Recognition," IEEE Transactions on Speech and Audio Processing, 1993, vol. 1, no. 3, pp. 334-344, 1991.

[6],[7] [Cucu, 2011] Cucu, H., "Towards a speaker-independent, large-vocabulary continuous speech recognition system for Romanian," Teză de doctorat, Universitatea Politehnica din Bucureşti, România, 2011

[8] http://en.wikipedia.org/wiki/Language_model

[10] http://en.wikipedia.org/wiki/Java_(programming_language)

[11] https://www.java.com/en/about/

[12] "The Java Language Environment". *1.2 Design Goals of the Java™ Programming Language*. Oracle.

[13] http://en.wikipedia.org/wiki/Garbage_collection_(computer_science)

[14] http://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html

[15] http://searchsoa.techtarget.com/definition/Java-virtual-machine

[16] http://www.cl.cam.ac.uk/teaching/1112/OOProg/Files/OOPLecture8.pdf

[17] Jelovic, Dejan. "Why Java Will Always Be Slower than C++". www.jelovic.com

[18] http://cmusphinx.sourceforge.net/wiki/ - cmusphinx_wiki

[19] http://www.merl.com/publications/docs/TR2003-110.pdf

[20] http://en.wikipedia.org/wiki/Java_API_for_RESTful_Web_Services

[21] https://jersey.java.net

[22] http://maven.apache.org/index.html

[23] http://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[24] http://en.wikipedia.org/wiki/Server-sent_events

[25] http://www.w3schools.com/html/html5_serversentevents.asp

[26] http://www.javaworld.com/article/2079190/scripting-jvm-languages/6-things-you-should-know-about-node-js.html

[27] http://www.mongodb.org/about/introduction/