UNIVERSITATEA POLITEHNICĂ DIN BUCUREŞTI
FACULTATEA DE ELECTRONICĂ, TELECOMUNICAŢII ŞI TEHNOLOGIA INFORMAŢIEI

PROIECT DE DIPLOMĂ

SERVICIU WEB DE RESTAURARE A DIACRITICELOR

Prezentată ca cerinţă parţială pentru obţinerea
titlului de *Inginer*
în domeniul *Electronică, Telecomunicaţii şi Tehnologia Informaţiei*
programul de studii *Reţele şi Software de Telecomunicaţii*

Profesori Coordonator:
Ş.L. Dr. Ing.Horia Cucu
Prof. Dr. Ing. Corneliu Burileanu

Student:
Ana-Maria Vladu

Bucureşti
2015

ANEXA 1

DECLARAŢIE DE ONESTITATE ACADEMICĂ

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

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

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

Bucureşti, *Data* Ana-Maria Vladu

CUPRINS

LISTA FIGURILOR

LISTA TABELELOR

LISTA ACRONIMELOR

NLP – Natural Language Processing/ Procesarea Limbajului Natural

PPL – Perplexity/ Perplexitate

OOV – Out of vocabulary/Cuvinte din afara vocabularului

MSD – morphological and syntactic descriptor/ Descriptor morfo-sintactic

LEX – Lexicon/ Dicționar

HTTP – HyperText Markup Language/ Limbaj de Marcare HiperText

PPL – Entropie Încrucișată

R – Corpus de Referintă

TT – Text Marcat Ideal

RT – Text Marcat dar Neetichetat

ML – Machine Learning/ Învățarea pe bază de Model a Programelor

URL – Uniform Resource Locator/ Locator Uniform al Resurselor

ASCII – American Standard Code for Information Interchange/ Codul Standard American pentru Schimbul de Informație

JVM – Java Virtual Machine/ Mașină Virtuală Java

CSJS – JavaScript pe Partea de Client/ Client Side JavaScript

DOM – Document Object Model/ Prototipul Obiectului Document

REST – Representational State Transfer/ Reprezentarea Stărilor de Transfer

JSP – Java Server Pages/ Pagini Java pe partea de Server

XML – Extensible Markup Language/ Limbaj de marcare extinsibil

INTRODUCERE

„*Limba română la sine acasă e o împărăţie bogată, căreia multe popoare i-au plătit banii în aur. A o dezbrăca de averile pe care ea le-a adunat în mai bine de 1000 de ani, înseamnă a face din împărătească cerşetoare...*”

Mihai Eminescu

1.1 MOTIVAŢIA LUCRĂRII DE DIPLOMĂ

Scrierea fără diacritice poate duce la exprimări ambigue, vulgare şi/sau cu un sens şi cu un conţinut total diferit care, în unele cazuri, poate genera grave ambiguități.

Un semn diacritic este un semn tipografic adăugat la o literă pentru a indica o diferenţă în pronunţie sau pentru a deosebi sensurile a două cuvinte altfel scrise identic. În limba română, semnele diacritice se plasează de obicei deasupra sau dedesubtul unei litere. Se folosc cinci litere cu semne diacritice: ă, â, î, ş, ţ.

Mai precis, semnele diacritice sunt Ă, Â, Î, Ș, Ț şi perechile lor minuscule, respectiv ă, â, î, ș, ț. Caracterele Ș şi Ț folosite corect sunt cele cu virgule. Aspectul şi denumirea acestor semne sunt:

Ă ă – căciulă (breve); când semnul este pus deasupra unei litere. Â â Î î – circumflex. Ș ș Ț ț – virguliţă sau virgulă, plasată sub literele corespunzătoare s, S, t, T.

În baza legilor în vigoare („Legea nr. 500/2004 privind folosirea limbii române în locuri, relaţii şi instituţii publice” şi „Legea nr. 24/2000, republicată, privind normele de tehnică legislativă pentru elaborarea actelor normative”), textul materialelor care emană de la instituţiile statului (şi cu atât mai mult a celor de utilitate publică) este obligatoriu să fie scrie corect, cu toate semnale diacritice necesare, atât în limba română, cât şi atunci când se reproduc cuvinte din alte limbi.

Scrierea fără diacritice poate conduce şi la cuvinte care nu există în limba română, precum „pamant”, „stiinta”, „samanta”, rezultate din scrierea incorectă, fără diacritice, a cuvintelor „pământ”, „ştiinţă”, „sămânţă”.

În alte limbi (franceză, germană, maghiară etc.), prin lege se dispune, clar şi la modul imperativ, că *scrierea fără diacritice este inacceptabilă*. În frecvente cazuri, textele scrise fără diacritice obligă numeroşi utilizatori la efortul de a introduce diacriticele ulterior, lucru migălos, care consumă mult timp şi nervi şi care generează nemulţumire.

Descoperirea unei modalităţi de a restaura diacriticele în mod automat este bine venită nu numai pentru texte vechi şi valoroase păstrate în format electronic, dar şi pentru texte contemporane, aceste din urmă continuând să fie redactate fără diacritice. Motivaţia unei astfel de scrieri (fără diacritice) este cauzată de lipsa tastelor standardizate. Factorii ergonomici pot fi, de asemenea, menţionţi (în cazul în care o persoană e nevoită să apese mai mult de două taste pentru a obţine un caracter cu semne diacritice, atunci, de cele mai multe ori în comunicaţia informală, ca de exemplu e-mail, el/ea va lua, probabil, cea mai simplă soluţie cu apăsarea unui singur buton).

Creşterea continuă a numărului de texte disponibile prin Internet face că metodele automate de inserare a diacriticelor să devină o componentă esenţială în multe aplicaţii importante, cum ar fi extragerea de informaţii, traducerea automată, colecţionarea de texte, construirea dicţionarelor electronice şi multe altele. Corectarea erorilor ortografice poate să aibă un impact major asupra calităţii rezultatelor obţinute în aceste aplicaţii. De exemplu, în absenţa unei metode de restaurare a diacriticelor, unele cuvinte devin ambigue, cum este cazul cuvintelor din limba româna peste, peşte sau paturi, pături. O căutare bazată pe astfel cuvinte poate returna multe texte irelevante (de exemplu, o căutare pentru peste ar returna şi documente conţinând peşte). De asemenea, traducerea unor astfel de cuvinte într-o limbă străină poate fi eronată (de exemplu, traducerea corectă a cuvântului „pături” în limba engleză este „blankets”, dar în absenţa diacriticei este tradus greşit ca şi „beds”).

În momentul de față există mai multe aplicații software care restaurează automat diacriticele. Unul dintre ele, foarte utilizat, este site-ul diacritice.com care folosește același software ca și programul disponibil pentru Windows: AutoCorect. Principiile pe care se ghidează nu sunt disponibile utilizatorilor, el este privit ca o cutie neagră care își face treaba cu o anumită precizie. Așa cum s-a menționat, procesul restaurării diacriticelor este necesar și în alte aplicații de sine stătătoare.

Lucrarea de față propune un serviciu web care poate fi accesat online de orice altă componentă software prin protocolul HTTP. Se elimină, astfel, dependența de sistemul de operare și se adaugă avantajul de a putea fi „moștenit” de către alte sisteme.

Un exemplu sugestiv este „Serviciul de analiză statistică a presei scrise online MediaAnalytics” . Acesta conține o bază de date extrasă din diverse articole din presa disponibilă pe internet, pentru ca procesul de analiză statistică să poată găsi doar articolele vizate, nu și articole fără relevanță este nevoie de o inserție a diacriticelor în toată baza de date.

Pentru a putea ilustra funcționalitatea sistemului s-a creat și o aplicație web cu scop demostrativ. Aceasta poate prelua de la utilizator textul în format .txt, sau direct de la tastatură. Se apasă pe

butonul care face o cerere către sistemul de restaurare, apoi rezultatul se poate observa pe pagina web, într-o altă zonă de text, creată special pentru primire rezultatului.

Sunt mai multe probleme cu care s-ar confrunta un astfel de program. Se poate menționa faptul că formatul textului nu este păstrat, în cele mai multe astfel de servicii disponibile pe internet. Dacă se va încarca în sistem o lucrare științifică cu diverse formate, formule, adnotații acestea nu vor fi apărea în fișierul rezultat. De exemplu pentru resturarea textului: „se vor constitui datele de intrare pentru algoritmul de dezambiguare[79]”, rezultatul va fi, de cele mai multe ori următorul text de ieșire: „se vor constitui datele de intrare pentru algoritmul de dezambiguare79”. Dacă este un text amplu, cum ar fi o lucrare de licență, astfel de erori vor fi numeroase, iar corectarea acestora nu se poate face decât manual.

O altă problemă importantă este mărimea textului și formatul de salvare: .doc, .docx, .txt. etc. Există, de asemenea, o limită de caractere ce poate fi încărcată maxim în sistemele disponibile pe internet, astfel că textul trebuie încărcat pe rând, iar pentru o intrare de cateva zeci de pagini, o astfel de procesare din partea utilizatorului este cât se poate de neplăcută. O altă chestiune care necesită lămurire este faptul că, uneori, nu se lucrează cu texte complet lipsite de diacritice, acestea mai apar ocazional, iar în momentul când sunt încărcate în sistem, codurile lor ASCII nu sunt recunoscute și sunt înlocuite cu anumite caractere bizare, care fac restaurarea diacriticelor un procedeu în care se câștigă timp pe o parte, dar se pierde pe alta. Motivul pentru care se întâmplă această transformare este că, deși toate caracterele cu semne diacritice sunt prezente în tablelul cu codurile ASCII, orice filtrare de 7 biți a unui text ce conține caracterele respective va fi corupt. Situația este și mai gravă atunci când aceste caractere nu se află în tabelul cu codurile ASCII. Un exemplu concludent este un astfel de text „...Practic, metoda propusă încearcă să învețe reguli aplicabile la nivel de literă” care produce următorul rezultat: „...Practic, metoda propusă A®ncearca să A®nvete reguli aplicabile la nivel de literă”.

Ceea ce se mai poate lua în calcul în momentul construcției unui astfel de sistem de restaurare este să se decidă asupra algoritmului de procesare a limbajului natural. Există mai multe metode de a restaura diacriticele, unele consumatoare de timp și de resurse, altele având o eroare mai mare. Aici intră în discuție eficiența inserției diacriticelor, pentru a putea alege modelul de limbă care simulează cel mai bine realitatea lingvistică. Perfecțiunea nu este de atins pentru un astfel de sistem, așa cum nu se poate atinge nici în realitate. Un astfel de model nu poate să rețină toate cuvintele din vocabularul limbii române și, mai mult, diferite conexiuni între cuvinte, pentru a putea să decidă când să insereze diacriticele și când nu. Un astfel de model se bazează pe procesarea limbajului natural (NLP), pe baza unor reguli învățate din experiență. Un astfel de sistem este antrenat cu corpuri de texte corecte, iar pe baza modelului creat, oferă soluții la probleme, respectiv situații, cu care nu s-a mai întâlnit la antrenare. Date fiind problemele cu care se confruntă un astfel de sistem, doar unele dintre ele au fost rezolvate cu succes de către serviciul web prezentat în teza de față, altele necesită o abordare ulterioară.

1.2 Obiectivele Proiectului de Diplomă

Teza de față își propune să restaureze diacriticele folosind un model de limbă ce se bazează pe pe n-grame, o noțiune care va fi explicată în detaliu în primul capitol al acestei teze. Această metodă pornește de la ideea că limba română este dependentă de context, iar o tehnologie bazată pe n-grame va fi extrem de utilă și rapidă și necesită un efort de dezvoltare redus. Metoda aleasă în procesul de restaurare folosește modele de limbă și o hartă a corespondențelor dintre cuvinte. Trebuie menționat faptul că metoda folosită este cea mai eficientă la ora actuală având o precizie foarte mare: erorile la nivel de cuvânt vor lua o proporție de 1,99%, iar erorile la nivel de caracter vor fi prezente în proporție de 0,48%.

În Capitolul 2 se vor sublinia și alți algoritmi de restaurare a diacriticelor, pentru fiecare, precizându-se eficiența, resursele necesare și timpul de prelucrare. În afara metodei pe care se

bazează teza de față, se vor descrie încă două metode și anume: metoda marcajelor și metoda inserării diacriticelor la nivel de caracter.

Teza va continua cu o prezentare a resurselor necesare pentru a creea un model de limbă adecvat și va aborda probleme precum procesarea limbajului natural și învățarea modelelor de limbă. Pe baza metodelor menționate anterior se vor atinge următoarele aspecte cu care se confruntă un sistem de resturare a diacriticelor: dicţionarele electronice nu sunt disponibile, sau doar dicţionare de dimensiuni relativ mici sunt făcute publice. Mai mult decât atât, în cazul în care dicţionarul însuşi nu are diacritice, metodele care se bazează pe aceasta resursă pentru restaurarea diacriticelor devin inaplicabile. Procesoarele folosite pentru analiza morfologică şi/sau sintactică, considerate folositoare pentru problema restaurării diacriticelor, nu există sau nu sunt public disponibile. Numărul de texte disponibile conţinând diacritice este relativ mic. Mărimea corpusurilor publice sau disponibile prin internet influenţează mărimea vocabularului care poate fi construit ad-hoc pe baza acestor texte.

Capitolul 3 se va ocupa cu tehnologiile de prelucrare a limbajului natural, a procesării textului și a tehnogiilor de programare. Se va acorda o atenție deosebită protocoalelor de comunicație, tehnologiilor web și a serverelor de aplicații.

Capitolul 4 se ocupă cu precădere asupra funcționalității serviciului web de restaurare a diacriticelor și se vor prezenta diverse scenarii de utilizare.

În capitolul concluzii se vor prezenta rezolvările abordate până în prezent de către serviciul web care face obiectul tezei de față și se vor enumera și aspectele care vor fi dezvoltate în viitor pentru îmbunătățirea aplicației și lărgirea gamei de utilizare.

CAPITOLUL 1 NOŢIUNI TEORETICE. GENERALITĂŢI

1.1 DIACRITICELE ÎN LIMBA ROMÂNĂ

Limba română este o limbă care foloseşte intens diacritice. Chiar dacă există doar 5 caractere cu diacritice (ă, â, î, ş, ţ), frecvenţa apariţiilor acestora este foarte mare: aproximativ 30-40% dintre cuvinte într-un text obişnuit sunt scrise cu diacritice. Un text scris fără diacritice, are, în general, aceste caractere substituite cu formele lor lipsite de diacritice: a, a, i, s, şi respectiv t. Cuvintele din limba română pot fi grupate în două categorii, după criteriul ambuităţii cuvintelor lipsite de diacritice.

- Cuvinte care *nu sunt ambigue* (cuvinte care fie nu au diacritice, fie au un tipar al diacritice unice): alb, astfel, pădure, ştiinţific.
- Cuvinte ambigue (cuvinte care pot fi scrise după mai multe tipare ale diacriticelor): casa/casă, până/pană/pana, bulgari/bulgări, ţări/ţari/tari/târî

Propoziţii fără diacritice	**Tiparul 1 al diacriticelor**	**Tiparul 2 al diacriticelor**
Tancul are 3 ani.	Ţâncul are trei ani.	Tancul are trei ani.
Am vazut o fata frumoasa.	Am văzut o faţă frumoasă.	Am vazut o fată frumoasă.

Romanul s-a nascut la Roma.	Românul s-a născut la Roma.	Romanul s-a născut la Roma.
Vrem zece paturi.	Vrem zece pături.	Vrem zece paturi.
Suporterii arunca cu bulgari.	Suporterii aruncă cu bulgări.	Suporterii aruncă cu bulgari.
Sa-mi dai pana maine.	Să-mi dai până mâine.	Să-mi dai pana mâine.
Tarii au decis.	Ţarii au decis.	Tarii au decis.

Tab. 1 Tipare ale cuvintelor

În general, lipsa diacriticelor într-un text din limba română poate cauza diferite probleme: lizibilitate scăzută, ambiguitate aparentă şi câteodată ambiguitate care nu are rezolvare. Chiar dacă la prima vedere nu ar părea important, lizibilitatea scăzută este un factor important când cititorul are nevoie să înţeleagă rapid un text în limba română. Dacă diacriticele lipsesc, anumite propoziţii (mai ales acelea în care procentul diacriticelor este mult peste medie) ar trebui citite de cel puţin două ori pentru a rezolva problema ambiguităţii aparente. Acest lucru este posibil doar în cazul în care textul este suficient de extins şi ambiguităţile pot fi rezolvate pe baza contextului. Dacă textul nu este suficient de larg, sau dacă doar câteva cuvinte sunt disponibile cititorului, atunci rezolvarea ambiguităţilor este mult mai dificilă sau chiar imposibilă. Câteva astfel de ambiguităţi sunt prezentate în tabelul de mai sus.

Numeroase corpusuri de text care vizează articolele de ştiri apar pe internet fără diacritice. În aceste tipuri de articole de ziare lizibilitatea este afectată semnificativ (ambiguitatea aparentă face ca textele să fie greu de citit), dar ambiguitatea nerezolvabilă apare rar deoarece cititorul are acces la tot contextul.

1.2 DESCRIEREA METODEI

Acestă metodă a fost implementată şi dezvoltată de *Lucian Petrică, Horia Cucu, Andi Buzo, şi Corneliu Burileanu*, fiind descrisă în articolul „A Robust Diacritics Restoration System using Unreliable Raw Text Data”. Ea se bazează pe modele de limbă statistice alcătuite din N-grame, sarcina restaurării diacriticelor este privită în teza de față ca un proces de dezambiguizare.

În general, un proces de dezambiguizare îşi propune să transforme un flux de etichete din vocabularul V_1 într-un flux corespondent de etichete din vocabularul V_2 conform hărţii probabilistice una-la-mai-multe. Această hartă specifică lista posibilităţilor de a transforma un cuvânt din V_1 în V_2 (plus probabilităţile lor asociate). Ambiguităţile în această hartă sunt rezolvate găsind secveţa din V_2 cu cea mai mare probabilitate posterioară dată în secveţa V_1.

Procesul restaurării diacriticelor ar trebui sa transforme un flux de cuvinte ambigue posibile (în cazul studiat este vorba despre cuvinte în care caracterele cu diacritice sunt înlocuite cu caracterele lor non-diacritice) într-un flux de cuvinte fără ambiguităţi (în cazul studiat cuvinte cu diacriticele corecte). Fiecare cuvânt w’, ambiguu din punct de vedere al poziţionării diacriticelor, din fluxul de date, este transformat într-un cuvânt cu diacritice estimat w^a, dată fiind secvenţa precedentă de N cuvinte cu diacritice corect poziţionate din **W**, prin găsirea unei forme cu diacritice a cuvântului w_i care maximizează formula:

$$\hat{w} = \arg_{w_i} \max p(w_i|W) \times p(w_i|w') \quad \textbf{(1.1)}$$

Prima probabilitate din ecuație este dată de modelul de limbă format din n-grame, în timp ce cealaltă este dată de harta cuvintelor cu corespondențe una-la-mai-multe. Dacă cuvântul fără diacritice w' nu este găsit în harta cuvintelor, atunci acest cuvânt este pur și simplu copiat în fluxul cuvintelor cu diacritice. (w^a=w').

Modelul de limbă bazat pe n-grame poate fi construit folosind un corpus de text ce conține cuvinte cu diacriticele poziționate corecte. Același corpus poate fi folosit și pentru estimarea probabilităților pentru harta corespondențelor dintre cuvinte (folosind numărul aparițiilor cuvintelor cu diacritice) cu ajutorul formulei (unde w_j sunt toate cuvintele posibile cu diacritice a cuvântului w'):

$$\hat{w} = \arg_{w_i} \max p(w_i|W) \times p(w_i|w') \quad \textbf{(1.2)}$$

$$p(w_i|w') = \frac{Număr\ apariţii\ (w_i)}{\sum_j Număr\ apariţii\ (w_j)} \quad \textbf{(1.3)}$$

Un exemplu de hartă probabilistică a corespondențelor între cuvinte se poate observa în Tab. 2 Harta Cuvintelor

Fragment dintr-o hartă probabilistică cu corespondențe una-la-mai-multe

...

dacia: dacia 1,0

fabricand: fabricând 1,0

pana: pana 0,005 pană 0,008 până 0,987

sarmana: sărmana 0,847 sărmană 0,153

tari: tari 0,047 ţari 0,002 ţări 0,942 târî 0,008

...

Tab. 2 Harta Cuvintelor

1.3 MODELE DE LIMBĂ STATISTICE BAZATE PE N-GRAME

Rolul unui model de limbă este să estimeze care este probabilitatea ca o secvenţă de cuvinte $W = w_1, w_2,..., w_n$, să fie validă pentru limba de interes (sursă). Luând ca exemplu următoarele două secvenţe: „Radacinile invataturii sunt amare dar fructele ei sunt dulci" şi „Rădăcinile învăţăturii sunt amare dar fructele ei sunt dulci", acestea pot fi interpretare corect de către un vorbitor român, dar în prima secvență, primele două cuvinte nu există în vocabular. Rolul unui model de limbă este de a atribui o probabiliate semnificativ mai mare celei de-a doua secvenţe.

Probabilitatea unei secvenţe de cuvinte $W = w_1, w_2, ..., w_n$ poate fi descompusă în următorul mod:

$$P(W)=p(w_1, w_2, \ldots, w_n)=p(w_1)p(w_2|w_1)\ldots p(w_n,| w_1, w_2, \ldots, w_{n-1}) \quad \textbf{(1.4)}$$

Aceasta înseamnă că atribuţia de a estima probabilitatea unei secvenţe de cuvinte W este împărţită în mai multe sarcini de estimare a probabilităţii unui singur cuvânt dată fiind o istorie a cuvintelor care îl preced. Din motive de calcul, istoria cuvintelor precedente nu se poate extinde pentru a include un număr infinit de cuvinte şi este limitată la un număr de *m* cuvinte. Altfel spus, se poate face presupunerea că doar un număr limitat de cuvinte influienţează probabilitatea următorului cuvânt. Această afirmaţie conduce la necesitatea definiri unui model de limbă bazat pe n-grame. Trigramele sunt folosite cel mai frecvent. Aceste consideră o istorie formată din două cuvinte pentru a putea prezice al treilea cuvânt. Această metodă necesită o colecţie de date statistice alcătuită din secvenţe de trei cuvinte, aşa numitele 3-grame (trigrame). Modelele de limbă pot fi estimate folosind 2-grame (bigrame), un singur cuvânt (unigrame), sau orice alt ordin de n-grame.

1.4 CONSTUCŢIA MODELELOR DE N-GRAME

Un model de limbă bazat pe n-grame este construit estimând probabiblităţile discutate mai sus, utilizând un corpus de text foarte mare. De exemplu, în cazul modelelor de limbă bazate pe bigrame, probabilităţile $p(w_i|w_j)$ pentru fiecare pereche de cuvinte (w_i, w_j) trebuie estimate. Pentru a calcula acestă probabilitate, se foloseşte principiul probabilităţii maxime (PM) şi se calculează cât de des cuvântul w_i este urmat de w_j şi nu de alte cuvinte.

$$p(w_j, w_i) = \frac{\text{Număr apariţii } (w_i, w_j)}{\sum_w Număr\ apariţii(w_i, w)} \quad \textbf{(1.5)}$$

Pentru un model de limbă bazat pe trigrame este nevoie de estimarea tuturor probabilităţilor

$$p(w_k\ |w_i, w_j) = \frac{Număr\ apariţii\ (w_i, w_j, w_k)}{\sum_w Număr\ apariţii\ \ (w_i, w_j, w)} \quad \textbf{(1.6)}$$

O cantitate mare de date pentru antrenare (în mod uzual milioane chiar miliarde de cuvinte) sunt necesare pentru a estima cu acurateţe aceste probabilităţi. De asemenea, n-gramele de ordin mai mare necesită o cantitate mai mare de date de antrenare. Problema perplexităţii, care este tipică pentru orice sistem statistic, trebuie luată în considerare şi este abordată în secţiunea următoare.

1.5 ABORDAREA PROBLEMEI PERPLEXITĂŢII DATELOR

1.5.1 METODE DE NETEZIRE

Una dintre problemele cheie ale modelării n-gramelor este perplexitatea intrinsecă a corpusului real de text folosit pentru antrenare. Fără a lua în considerare cât de mare este corpusul de antrenare, o să existe n-grame care nu apar la antrenare, dar care îşi vor face apariţia la evaluare, sau în corpusul de text. În aceste cazuri extreme, probabilităţile atribuite unor n-grame care nu apar, având în vedere estimarea probabilităţii maxime, sunt 0. Mai sunt cazuri în care n-gramele

apar de puţine ori (mai puţin de 10 apariţii) în corpusul de antrenare. Mai mult, problema devine mai severă pe măsură ce se folosesc n-grame de ordin superior. În toate aceste cazuri, probabilităţile ce au fost estimate după numărul apariţiilor în corpusul de text de antreare, sunt estimate grosolan şi necesită o adaptare.

Medolele implicate în acest proces de adaptare sunt numite metode de netezire. Acestea scad un procent din probabilităţile n-gramelor care apar în funcţia probabilităţilor cumulate şi o redistribuie n-gramelor care nu şi-au făcu apariţia la antrenare. Există mai multe metode de netezire care tind să particularizeze redistribuirea probabilităţilor în funcţia de probabilităţi cumulate, date fiind anumite motive specifice.

O metodă de netezire de bază se numeşte *netezire prin adăugare-lui-1*. Acesta pur şi simplu adaugă un număr fix (de exemplu 1) la fiecare numărare a n-gramelor. Asta înseamnă că n-gramelor care nu au apărut în corpusul de antrenare, dar sunt alcătuite din cuvinte din vocabular, li se vor atribui probabilităţi nenule. Analizând probabilităţile nou atribuite, se remarcă rapid că netezirea prin adaugarea lui unu dă o credibilitate nejustificată unor n-grame ce nu apar în corpusul de formare. Un remediu simplu ar fi să se adauge un număr mai mic α, în loc de 1 (metoda se numeşte *netezire prin adăugarea lui α*). Acest numărul, α, trebuie să fie empiric estimat într-un corpus mare de text.

Netezirea prin eliminarea interpolării încearcă să ajusteze numărul de apariţii ale n-gramelor răspunzând la întrebarea: „Dacă observăm că o n-gramă de c ori în corpusul de formare, cât de des ne aşteptăm să o vedem într-o aplicaţie reală (ca de exemplu în corpusul de test)?”. Metoda împarte corpusul de formare în două parţi şi foloseşte o parte pentru a estima apariţiile n-gramelor şi a doua pentru a răspunde la întrebarea de mai sus. În al doilea rând, schimbând rolurile celor două părţi şi interpolând rezultatele, metoda duce la obţinerea unor rezultate cu aproximări mai bune decât metoda cu prin adăugarea lui α.

O altă metodă de netezire, *Good-Turing,* foloseşte apariţiile reale (c) şi ţine evidenţa numărului apariţiilor (N_c este numărul n-gramelor care apar de c ori în corpusul de formare) pentru a adapta numărul apariţiilor (c^*) pentru toate n-gramele prezente şi absente.

$$c^* = (c+1)\frac{N_{c+1}}{N_c} \tag{1.7}$$

Metoda Good-Turing furnizează o metodă bazată pe nişte reguli clare pentru a adapta numărul apariţiilor, dar nu este de încredere pentru un c foarte mare, pentru care N_c este tipic 0. Acest dezavantaj poate fi rezolvat dacă nu se aplică adaptarea pentru n-grame frecvente.

1.5.2 METODELE DE BACK-OFF

O abordare secundară pentru a rezolva perplexitatea datelor este folosirea mai multor modele de limbă, cu avantajul particular de a crea un model de limbă interpolat care poate beneficia de toate părţile sale constitutive. De exemplu, n-gramele de ordin mai mare pot furniza un context adiţional valoros, dar n-gramele de ordin inferior sunt mai robuste. Dacă pentru mai multe ordine de n-grame (1, 2 şi 3) p_n este deja contruit un model de limbă interpolat p_I poate fi construit prin combinaţia lor liniară.

$$p(w_3|w_1,w_2) = \lambda_1 p_1(w_3) \times \lambda_2 p_2(w_3|w_2) \times \lambda_3 p_3(w_3|w_1,w_2) \tag{1.8}$$

Coeficienţii λ din ecuaţia de mai sus trebuie să fie pozitivi, numere subunitare şi suma acestora este 1. În funcţie de raportul lor, LM-urile de ordin mai mic, sau cele de ordin superior devin mai exacte. Valorile coeficieţilor se stabilesc empiric pentru un set extins.

Mecanismul de back-off utilizează LM-uri bazate pe n-grame interpolate pentru a întâmpina problema n-gramele absente printr-o metodă uşor diferită faţă de metode netezirii. Dacă dorim să estimăm probabilitatea unei n-grame care nu apare, sau apare foarte rar în corpusul de antrenare, o idee bună ar fi să se ia în considerare probabilitatea atribuită n-gramelor de ordin mai mic. Optimizarea constă în alegerea echilibrului corect între modelele de ordin superior şi toate celelalte modele de ordin inferior (dacă vor fi folosite vreodată). Mai multe metode de back-off au fost propuse începând cu metoda de *netezire a lui Witten-Bell*, care se concentrează pe diversitatea cuvintelor care urmează după o istorie. Metoda cea mai generală folosită astăzi este metoda de netezire a lui *Kneser-Ney* introdusă în, care ia în cosiderare diversitatea istoriei unei n-grame particulare. O extensie a acestei metode este metoda modificată de netezire a lui Kneser-Ney, care foloseşte o metodă numită actualizarea absolută pentru a reduce funcţia probabilităţilor cumultate pentru evenimente care şi-au făcut apariţia.

Pentru un corpus analizat în particular, metoda modificată a lui Kneser-Ney conduce la o perplexitate cu 5-10% mai scăzută faţă de orice alte metode .

1.6 METRICI DE EVALUARE ALE MODELELOR DE LIMBĂ

Rolul unui model de limbă este să prezică următorul cuvânt daţi fiind predecesorii săi, profitând de avantajul redundanţei limbii. Capabilitatea de predicţie poate fi măsurată obiectiv dacă se doreşte o comparaţie între modele de limbă diferite sau îmbunătăţirea acestora.

PERPLEXITATEA

Metrica cea mai generală pentru evaluarea unui model de limbă este *eroarea la nivel de cuvânt.* Puterea de predicţie a unui model de limbă constă în măsurarea probabilităţilor acordate modelului de limbă pentru a testa o secvenţă de cuvinte. Un model de limbă bun ar trebui să atribuie o probabilitate mare unui text corect şi o probabilitate scăzută unui text mai prost. În cazul celei mai generale metrici de evaluare este perplexitatea. Perplexitatea derivă din entropia încrucişată, o mărime ce poate fi calculată dat fiind un model de limbă particular LM şi o secvenţă de cuvinte particulară $W=w_1, w_2, ..., w_n$, aşa cum urmează.

$$H(p_{1,M}) = -\frac{1}{n}\log p_{1,M}(w_1, w_2, \dots, w_n) = -\frac{1}{n}\sum_{i-1}^{n}\log p_{1,M}\,(w_i|\, w_1, w_2, \dots, w_{i-1}) \quad \textbf{(1.9)}$$

Perplexitatea derivă din entropia încrucişată, utilizând o transformare simplă:

$$PPL(p_{1,M}) = 2^{H(p_{1,M})} \quad \textbf{(1.10)}$$

O mai mare perplexitate pentru o secvenţă de cuvinte înseamnă o capacitate mai mică de predicţie pentru modelul de limbă, dată fiind o secvenţăde cuvinte particulară. De fapt, perplexitatea poate fi calculată atât pe un *set pentru testare* cât şi pe un *set de antrenare* şi, evident, are o semnificaţie uşor diferită pentru cele două cazuri. Perplexitatea în cazul setului de test evaluează generalizarea şi predicţia capabilităţilor unui model de limbă, în timp perplexitatea setului de formare măsoară cât de bine se potriveşte modelul de limbă cu datele de antrenare.

CUVINTE DIN AFARA VOCABULARULUI

Toate metodele de netezire descrise mai sus se confruntă cu n-grame care nu fac parte din corpusul de antrenare, dar sunt formate din cuvinte care apar în corpusul de antrenare. Aceste n-grame nu pot fi folosite în procesul de adaptare pentru atribuirea probabilităţilor nenule unui cuvânt care nu face iniţial parte din vacabular. Aceste cuvinte sunt numite *cuvinte din afara vocabularului (OOV)* şi în consecinţă nu pot fi prezise de către un model de limbă.

Aceste cuvinte OOV îngreunează procesul de evaluare. Deoarece perplexitatea lor este infinită, aceasta nu se poate adăuga la rezultatul perplexităţii unor alte n-grame pentru a obţine perplexitatea unui întregi secvenţe de cuvinte. În acest caz, în afara perplexităţii, procentul cuvintelor OOV (din numărul total de cuvinte) trebuie să fie specificat şi toate aceste metrici trebuie luate în considerare pentru comparaţie:

$$OOVuri[\%] = \frac{\#OOVuri}{\#cuvinte} \times 100 \quad \textbf{(1.11)}$$

APARIȚIILE N-GRAMELOR

Apariția n-gramelor este încă o metrică care poate fi folosită pentru trage câteva concluzii privind capabilitatea de predicţie a unui model de limbă format din n-grame. Aşa cum se observă în secţiunile anterioare, modele de back-off folosesc mai multe modele de limbă pentru abordarea problemei dispersiei datelor. De exemplu, un model de limbă construit din trigrame încercă să prezică cuvântul actual bazându-se pe o istorie alcătuită din două cuvinte (modelul trigramei), dar se poate întoarce (datorită datelor insuficiente) la o istorie alcătuită dintr-un singur cuvânt care precede cuvântul curent (modelul bigramei) sau nici macar nu se va lua deloc în considerere o istorie a cuvintelor (modelul unigramei). Pentru un model bazat pe o trigramă, procentul potrivirilor arată o măsură a numărului cazurilor în care modelul foloseşte complet istoria celor două cuvinte precedente în paralel cu numărul cazurilor în care modelul are nevoie să se întoarcă pentru a găsi probabilitatea n-gramei actuale:

$$Aparițiile\ trigramelor[\%] = \frac{\#concordanţele\ trigramelor}{\#cuvinte} \times 100 \quad \textbf{(1.12)}$$

Metrica bazată pe aparițiile n-gramelor, care poate fi folositoare pentru a compara modele de limbă specific unor domenii de interes. Cu cât procentul concordanţelor este mai mare cu atât modelul de limbă este mai bine adaptat domeniului de interes.

1.7 EXPERIMENTELE RESTAURĂRII DIACRITICELOR

Două corpusuri de text ce conțin cuvinte corecte din punct de vedere al diacriticelor se numesc europarl și misc. Primul conține informații din domeniul discuțiilor euro-parlamentare (225k propoziții și 5.3M cuvinte), iar cel de-al doilea conținele articole din ziare și literatură (400k propoziții și 11M cuvinte). Acestea au fost folosite pentru a construi un sistem de restaurare a diacriticelor (modelul de limbă și harta cuvintelor). De fapt, corpusul misc a fost mai întâi împărțit într-o parte mai mare (90%) folosită pentru antrenare și restul (de 10%) a fost folosită în procesul de evaluare.

Pentru a găsi cea mai bună organizare pentru sistemul de restaurare a diacriticelor, s-a variat modelul de limbă de la 2 la 5 n-grame și, de asemenea s-a folosit o hartă a cuvintelor. Utilitarul SRI-LM a fost folosit pentru crearea modellelor de limbă și, de asemenea pentru procesul de dezambiguizare (aplicația *disambig*) pentru un text de intrare fără diacritice.

Diversele versiuni ale sistemului au fost evaluate în termenii erorii la nivel de cuvânt (WER) și la nivel de caracter (ChER), asupra părții de evaluare a corpusului misc,folosind NIST (Utilitar de Evaluare a Recunoașterii Vocale). Valorile experimentale sunt prezentate în Tab. 3.

Exp	LM	WER(%)	ChER(%)
1	2-grame	2,07	0,50
2	3-grame	1,99	0,48
3	4-grame	1,99	0,48
4	5-grame	2,00	0,54

Tab. 3 Rezultatele Restaurării Diacriticelor

Așa cum se observă din Tab. 3 variațiile ordinelor n-gramelor nu aduc îmbunătățiri importante dacă la antrenare se folosește un corpus relativ mic (aproximativ 15M cuvinte, în cazul de față). Totuși, rezultatul obținut datorită folosirii hărții cuvintelor se îmbunătățește.

COMPARATII CU ȘI PRIVIRE LA ALTE METODE DE RESTAURARE A DIACRITICELOR

În afară de această metodă, există mai multe metode fundamental diferite pentru restaurarea diacriticelor în limba română.

O altă metodă mai elaborată va fi prezentată în capitolul următor care folosește analiza sintactică pentru a dezambiguiza diferile ipoteze de cuvinte cu diactritice. Totuși, această metodă dă rezultate mai slabe decât algoritmul propus: 2,25% WER și 0.60% ChER. Rezultatele au fost obținute pe un set de test diferit (nu există un corpus de evaluare standard pentru sistemul de restaurare a diacriticelor pentru limba română)

În concluzie, se poate afirma că, din toate sistemele disponibile, acesta este cel mai bun.

O ANALIZĂ MAI DETALIATĂ ASUPRA SISTEMULUI

În general, evaluarea sistemului de resturare a diacriticelor este făcută în termeni de WER și ChER. Totuși, aceste două performanțe nu sunt capabile să sublinieze capabilitatea sistemului de a restaura anumite caractere individuale ce conțin diacritice. O analiză mai amănunțitî, la nivel de caracter a trebuit să fie realizată pentru a obține mai multe detalii despre care caracter diacritic este mai bine restaurat și care dintre ele este cel mai vulnerabil la erori.

Pentru această evaluare, s-au folosit trei alte unități ale perfomanței: precizia, retragerea și mărimea-F. Precizia este un raport dintre numărul total de caractere corect inserate și numărul total de diacritice din textul ipotetic, în timp ce mărimea „retragerii” este un raport dintre numărul diacriticelor corect inserate și numărul diacriticelor din textul de referință. Mărimea-F este o medie armonică între precizie și „retragere”.

Clasa de ambiguitate	Caracterul	Precizia (%)	Mărimea-F
a/ă/â	a	98,28	97,99
	ă	94,42	95,27
	â	98,79	98,16
i/î	i	99,97	99,92
	î	99,26	99,45
s/ș	s	99,75	99,69
	ș	98,71	98,92
t/ț	t	99,52	99,57
	ț	97,74	97,47
Total	Total	98,73	98,73

Tab. 4 Evaluarea la nivel de character a sistemului de restaurare a diacriticelor

În Tab. 4 se infățișează mai multe metrici ale performației pentru caracterele individuale care sunt subiectul procesului de restaurare a diacriticelor. Pe baza acestor rezultate, prima concluzie care se poate trage este că metoda cunoaște o performanță mai bună pentru caracterele fără diacritice (a,i,s,t). De asemenea, anumite clase de ambiguitate (i/î; s/ș) sunt aproape rezolvate perfect în timp ce altele (a/ă/â) pun foarte multe probleme. Ambiguitatea a/ă este o problemă specifică și dificilă pentru limba română, deoarece toate substantivele feminine și adjectivele a căror formă neaccentuată se termină cu ă au și o formă accentuată care se sfârșește cu a. În consecință, aceste forme ale cuvintelor nu pot fi dezambiguizate cu ușurință și ar fi nevoie de un ordin mai mare al n-gramelor pentru modelul de limbă sau de niște metode lingvistice pentru abordarea ambiguității.

EXEMPLU PRACTIC DE FOLOSIRE A SISTEMUL DE RESTAURARE A DIACRITICELOR

Dezvoltarea sistemului de restaurare a diacriticelor a fost solicitată pentru prima oară pentru a corecta un corpus mare de teste colectate de pe Internet, cu scopul final de a crea un model general de limbă pentru sitemul RAV (Recunoștere Automată a Vorbirii) pentru limba română. Performața sistemului a fost evaluată, în consecință, în contextul RAV.

Pentru acest experiment, s-au folosit modele acustice bazate pe HMM-ri și pe un dicționar fonetic. Pentru modelul de limbă s-a folosit un corpus de știri general. În *exp1,* modelul de limbă a fost antrenat pe un corpus fără diacritice. În *exp*2, modelul de limbă a fost antrenat cu textul cu diacriticele restaurate prin utilizarea metodei descrise. În Tab. 5 se prezintă WER.

Exp	Restaurarea Diacriticelor	WER (%)
1	Fără restaurarea diacriticelor	64,50
2	Cu LM realizate prin metoda n-gramelor	29,70

Tab. 5 Restaurarea diacriticelor în contexul RAV

CAPITOLUL 2 ALTE METODE DE RESTAURARE A DIACRITICELOR

2.1 METODA MARCAJELOR

Acestă metodă a fost implementată și dezvoltată de Dan Tufiș și Adrian Chițu, fiind descrisă în articolul „Automatic Diacritics Insertion in Romanian Texts". Ea se bazează pe tehnologia marcării probabilistice. Cuvintele din limba română se împart, ca și în metoda descrisă în teza de față, în cuvinte fără amguități (cuvinte-U) și cuvinte ambigue (cuvinte-A).

Iată câteva exemple cu șiruri de caractere care nu reprezintă cuvinte pentru limba română:

A) padure (pădure), tufis (tufiş), cantar (cântar), carare (cărare), casmir (caşmir), macar (măcar), fara (fără), cati (câţi), etc.

Asemenea șiruri de caractere sunt numite cuvinte-U.

Pentru a exemplifica ambiguitatea cauzată de lipsa diacriticelor, se va considera șirul *fata.* Într-un text în care diacriticele au fost îndepărtate, șirul ar putea să țină locul următoarelor variante:

B) fata, fată, fâţa, fâţă, faţa, faţă, făta, fătă.

Toate aceste înlănțuiri de caractere de tipul *fata* de mai sus (care ar putea ţine locul mai multor cuvinte: atât lipsite, cât şi cu diacritice) vor fi referite care cuvinte-A, sau cuvinte ambigue.

Elementele care nu sunt nici Cuvinte-A nici Cuvinte-U sunt denumite simplu 'cuvinte'.

S-a ajuns la concluzia că informația morfo-sintactică **dezambiguizează** cele mai multe din cuvintele-A. Totuși, mai există un subset al cuvintelor ambigue pentru care descriptorii morfosintactici sunt identici, iar distincția restaurării diacriticelor poate fi făcută doar pe baza înțelesului:

C) fata (Ncfsry), fâţa (Ncfsry), faţa (Ncfsry).

Se vor prezenta în continuare etichetele MSD pentru substantive pentru a exemplifica modul cum au fost atribuite:

Atribute (ro)	Valoare (en/ro)	Codul (en/ro)
CATEGORIE	Noun/Substantiv	N
TIPUL	comun	c
	propriu	p
GEN	masculin	m
	feminin	f
NUMĂR	singular	s
	plural	p
CAZUL	vocativ	v
	direct	r
DEFINIT/INDEFINIT	nu	n
	da	y
ACCENTUARE	neaccentuat	n
	accentuat	y

Tab. 6 Specificările românești pentru substantiv

Aceste cuvinte, care se vor numi Cuvinte-S, necesită o dezambiguizare din punct de vedere al sensului. Cuvinte S sunt un subset al cuvintelor A.

În Tab. 7 se pot observa datele extrase din corpusul de referință pe care îl folosește sistemul DIAC^{+}. Corpusul jurnalistic conține articole din revista săptămânală „Agenda” din Timișoara (2003-2006). Corpusul datelor din mediul juridic este o colecție de 6000 documente în limba română extrase din Jrc-Acquis. Adnotațiile părților de vorbire au fost făcute cu programul de etichetare pentru a minimiza cât se poate de mult numărul erorilor de etichetare. Numărul total de cuvinte ilustrate în Tab. 7 (prima linie) nu includ numere sau etichete care conțin una sau mai multe cifre, nume proprii, cuvinte străine (etichetate cu eticheta X), abrevierile (etichetate cu Y), datele calendaristice (marcate cu eticheta DATE) și semnele de punctuație. Din numărul total de etichete din textele menționate, etichetele care nu sunt luate în seamă sunt 36% și 26% respectiv. Diferența mare între numărul de etichete eliminate din cele două corpusuri de text este datorată faptului că corpusul gazetăresc conține multe numere (la secțiunea sportivă se pot găsi scorul, minutul, duratele, distanțele), cuvinte străine și abrevieri. Aceste categorii nu sunt semnificative pentru problema restaurării diacriticelor deoarece, în marea majoritate a cazurilor, acestea nu conțin semne diacritice. Totuși, numele proprii din limba română sunt cuvinte care pot conține diacritice, deci sunt relevante în procesul inserției diacriticelor. Totuși în textele cu caracter juridic, cu toate că numele de persoane sunt frecvente, niciunul din ele nu conțin diacritice. În

concluzie, pentru a face o comparație care să aibă sens între cele două registre ale limbii, s-au exclus numele prorpii din analiza de față.

Sunt două tipuri diferite de Cuvinte-S, în funcție de ce set de etichete se folosește în etichetarea părților de vorbire (POS): un set redus (setul-C din linia 5) și descriptorii morfo-sintactici ai lexiconului (setul MSD din linia 6). Cele două coloane demonstrează că restaurarea diacriticelor este realizată cu mai multă acuratețe când sistemul are acces la un context de informație lingvistică mai amplu. Pe de altă parte, utilizând un set redus comparativ cu unul mai larg mărește acuratețea procesului de marcare, ceea ce este vital pentru abordarea restaurării diacriticelor. Setul-C și setul MSD reprezintă modalitatea de a rezolva tensiunea dintre cardinalitatea și acuratețea setului de etichete. Această problemă va fi discutată pe larg în cele ce urmează.

Corpus	***Jurnalistic***	***Juridic***
1. Cuvinte	6.680.448	3.511.093
1* Caractere	370.088.236	21.404.666
2. Cuvinte cu diacritice (din 1.)	2.004.763 (30,01%)	1.026.385 (29,23%)
2* Diacritice	2.351.220	1.192.875
3. Cuvinte-U (din 2.)	238.132 (11,88%)	175.822 (17,13%)
4. Cuvinte-A (din 2.)	1.766.631 (88,12%)	850.563 (82,87%)
5. Cuvinte-S (Setul-C, din 4.)	58.420 (3,31%)	38.323 (4,51%)
6. Cuvinte-S (Setul MSD, din 4.)	24.916 (1,41%)	16.463 (1,94%)

Tab. 7 Distribuția cuvintelor cu diacritice în diferite registre ale limbii

Așa cum se poate observa în tabelul de mai sus, în textele din limba română, aproape o treime din cuvinte conțin cel puțin un caracter diacritic (30% din textele din ziare conțin în medie 1,17 semne diacritice în timp ce 29% din textele cu caracter juridic conțin în medie 1,16 caractere cu diacritice). Din totalul cuvintelor ce conțin diacritice doar un procent mic sunt cuvinte-U (12% din textele jurnalistice și 17% din cele juridice). Acest procent înseamnă, în cazul ideal: cu un dicționar cu acoperire completă și un text fără erori de tipografie, 25% ($\frac{\#\text{Cuvinte} - \text{A}}{\#Cuvinte}$) din numărul total de cuvinte dintr-un text ce aparține vocabularului curent vor rămâne ambigue. Într-un cadru mai realistic, aceste date sunt fictive deoarece niciun dicționar nu oferă o acoperire completă și cele mai multe texte conțin greșeli de tipografie (și nu le lipsesc diacriticele). În textele disponibile sistemului de față s-au identificat 72.722 (1,09%) erori de tipografie, în textele jurnalistice și 29.387 (0,84%) greșeli de tipografie în textele cu caracter juridic. Mai jos sunt listate principalele categirii de erori:

a) Chiar dacă un cuvânt conține diacritice, se poate să nu le conțină pe toate (ex. „invățământ” și „învățământ”, „lacătuș” și „lăcătuș” etc.);

b) Chiar dacă un cuvânt conține diacritice, una sau mai multe ditre ele pot fi greșite (ex. „sărmă” și „sârmă”, „câtre” și „către”, „neâncăpător” și „neîncăpător” etc.)

c) Chiar dacă un cuvânt conține diacritice, ele se poate să nu fie în concordanță cu ortografia din vocabularul în uz („considerînd” și „considerând”, „curînd” și „curând” etc.)

d) Cuvintele (cu sau fără diacritice) scrise greșit (ex. „înopta” și „înnopta”, „indenmizație” și „indemnizație”, „compensdiu” și „compendiu” etc.) nu li se pot aplica marcaje (ex. „5%pentru” și „5% pentru” etc.)

e) O etichetă din vocabular poate fi deformată, recuperarea ei fiind dificilă.

Se poate afirma că un corector de cuvinte tradițional al unui editor de text poate detecta aceste erori, iar utilizatorul le-ar putea corecta, dar cel puțin pentru limba română, acest lucru nu este valabil în totalitate datorită cuvinte-A (cuvinte care rămân cuvinte valide în limba română chiar și după îndepărtarea diacriticelor ex. „peste” și „pește”, „scoala” și „școala”), „barca” și „barcă”. Un program de corecție tradițional nu ar putea detecta cuvinte-A problematice.

Soluția tradițională a programului de corecție, cuvintele „din afara vocabularului” sunt subliniate și se așteaptă de la utilizator să aleagă varianta corectă dintr-o listă de variante posibile (care poate să nu includă soluția corectă). În abordarea care se face de către cei doi autori, Dan Tufiș și Adrian Chițu, majoritatea corecțiilor (în afară de cuvintele-S) sunt automat efectuate fără ca utilizatorul să aibă vreo intervenție. În timp ce corecția automată este realizată într-un timp foarte scurt, procedura manuală poate conține erori, și chiar dacă ese folosește un corector de cuvinte tradițional, poate dura ore, zile sau chiar luni întregi pentru a restaura un corpus mare de text. Pentru cuvintele-S care apar în text, sistemul se comportă ca un corector ortografic, și cere utilizatorului să facă o alegere, din lista cuvintelor posibile. O corecție plauzibilă contextual ar trebui să se adapteze cu restricțiile lingvistice asociate cuvintelor-S respective. De exemplu, dacă un cuvânt-S „fata” a fost marcat ca un substantiv comun, formă singulară, atunci celelalte forme ale acestui cuvânt (fata, fața, fâța) sunt caracterizate de același atribut morfo-lexical ca și cuvântul original. Tote celelalte variante ((fată, faţă, fâţă,fâta, fătă) ar trebui să fie ignorate datorită descritorilor morfo-sintactici diferiți.

Ultimele două linii din Tab. 7 Distribuția cuvintelor cu diacritice în diferite registre ale limbiiarată unde se folosește un set de etichete de o granularitate mai fină (614 etichete în setul MSD și 92 etichete în setul-C), numărul cuvintelor-S (etichetele pentru care formele cu diacritice nu pot fi determinate în mod statistic) este de două ori mai mic decât înainte. Se poate remarca faptul că majoritatea cuvintelor-S, în acest caz, conține greșeli de ortografie autentice, foarte puține cuvinte necesită o dezambiguizare de sens.

În secțiunea următoare se vor descrie pe scurt tehnologiile care stau la baza sistemului de restaurare DIAC^{+}, se va furniza o evaluare și câteva detalii asupra implementării sale.

2.1.1 PREPROCESAREA TEXTULUI

De când DIAC^{+} a fost creat pentru a lucra cu documente în format MS, sistemul extrage datele textuale din fișierul de intrare și îl stochează într-un format intern, adecvat utiliatarelor de pre-procesare, utilizând o bază de date și un sistem de interogare complet – Lucene. Textul extras din fișierul de intrare este tipizat și etichetat, creându-se în consecință un spațiu de cunoștințe lingvistice pentru textul actual pentru care restaurarea diacriticelor va ave loc.

2.1.2 TOKENIZER-UL

Tokenizer-ul este un program care identifică în textul de intrare unităţile elementare de procesare numite unităţi lexicale. Un element lexical este corespunzător, în general, ideii de cuvânt şi anume o secvenă de caractere delimitate de spaţii albe. Cu toate acestea mai multe

cuvinte pot forma o unitate lexicală, naturală (cum este *pentru că*) sau dimpotrivă o secvenţă de caractere delimitate de spaţiu pot fi împărţite în unităţi lexicale distincte (*dă-mi-le*) .
Tokenizer-ul recunoaşte, de asemenea, ca un singur element datele calendaristice (1 ianuarie, 1999; 01/01/99), abrevierile (dl, ADN-ul, dr.), diferite tipuri de punctuaţie.

2.1.3 MARCAREA SECVENŢIALĂ

Pentru a codifica principalele proprietăţi morfo-lexicale este necesar un sistem mare de coduri de descriere. Proiectul european MULTEXT în coordonare cu Grupul EAGLES de specificare lexicală a dezvoltat un set de recomandări pentru limbile din Europa de Vest. Ţinând seama de aceste specificaţii, Proiectul MULTEXT-Est Copernicus a dezvoltat în continuare sistemul pentru alte şase limbi din Europa Centrală şi de Est şi a dezvoltat resurse lexicale voluminoase (Tufiş, 1998). Setul de descriptori morfo-sintactici (MSD) specific românesc cuprinde 615 coduri. În conformitate cu practica actuală şi în conformitate cu normele statistice acest număr este foarte mare. Cu cât este mai mare setul de etichete cu atât este nevoie de un corpus de text pentru antrenare mai larg.

Este bine cunoscut că, odată cu lărgirea setului de etichete trebuie să se pună la dispoziție și un corpus de antrenare mai mare și din păcate, între cele două nu este o dependență liniară. Pentru a evita dispersia datelor și degradarea acurateții, un volum mare de muncă manuală ar fi necesar pentru construirea unui corpus de antrenare adecvat.

Marcarea secvențială este o tehnică în două etape ce se concentrează pe problema dispersiei datelor. În general, marcarea secvențială folosește un set de etichete ascuns (numit set-C), de o dimensiune mai mică (în cazul nostru 92 de etichete) construit pe baza modelului de limbă LM. LM contribuie la primul nivel de etichetare. Apoi, faza a doua înlocuiește etichetele din setul redus cu cele mai probabile etichete din punct de vedere contextual din setul MSD, ce conține 615 etichete. Ideea fundamentală în abordarea marcării secvențiale este că valorile din câmpul MSD și formele cuvintelor nu sunt independente. Având vocabularul MSD, dintr-o formă anume a unui cuvânt și un subset de atribute, în cea mai mare majoritate a cazurilor, se pot deduce restul perechilor de valori caracteristice formei actuale a cuvântului. Această proprietate se numește *capacitate de recuperare MSD*. Subsetul atributelor din setul-MSD care are capacitate de recuperare reprezintă setul atributelor din care setul-C este format.

Cuvintele ce devin ambigue reprezintă un procent mic (mai puţin de 10%) şi sunt în continuare prelucrate prin intermediul unor reguli forte simple: se cercetează, în funcţie de clasa de ambiguitate aleasă, la stânga cuvântului şi la dreapta sa, sau în ambele direcţii, dar pe distanţă limitată (care nu depăşeşte niciodată 4 cuvinte într-o direcţie) pentru o etichetă pentru eliminarea ambiguităţii. Rata de succes a acestei de-a doua etape este de 98%, ceea ce înseamnă că eroarea introdusă de setul-C este mai mică de 0,2%.
Se va avea în vedere un cuvânt W, într-un text deja etichetat cu eticheta T (din setul-C), aparţinând unei clase de ambiguitate şi o listă de etichete MSD care sunt disponibile pentru T. Corespondența de la T la eticheta MSD ar trebuie să fie aproape deterministă. În acest caz singura cerință deterministă tradusă într-o corespondenţă a etichetelor este unică în 90% din cazuri.

Vocabularul care stă la baza inducției setului-C și care este suport abordării marcării secvențiale, conține cuvinte tipizate cu etichete MSD, o intrare având următoarea formă: <cuvânt> <lemă> <msd>. Pentru limba română, acest lexicon, se va numi LEX, conține 800.000 de intrări.

Pentru un număr mai mic al setului-C de etichete, procesul de recuperare poate face față unor ambiguități care vor fi rezolvate folosind resurse adiționale de informație. În (Tufiș 1999) această nouă resursă este un set de reguli de dezambiguizare contextuală scrisă de mână. Aplicabilitatea atât a regulilor deterministe cât și a recuperării bazate pe reguli clare este limitat

doar de cuvinte din setul MSD. S-a asociat faza a doua a procesului etichetării secvențiale cu recuperarea MSD bazată pe entropie (Ceaușu, 2006). După această abordare, regulile de conversie a etichetei-C în MSD sunt în mod automat învățate din corpus şi aplicabilitatea lor nu necesită căutarea efectivă în setul MSD. Prin urmare, etichetele-C atribuite cuvintelor necunoscute pot fi convertite în etichete MSD. Dacă un vocabular MSD este disponibil, înlocuirea etichetelor-C cu etichetele MSD adecvate este făcută cu o acuratețe de 100%.

2.1.4 ARHITECTURA GENERALĂ A PROGRAMULUI DE INSERŢIE AUTOMATĂ A DIACRITICELOR

După cum s-a sugerat deja în secţiunea anterioară, componentele de bază ale unui program de introducere automată a diacriticelor sunt un dicţionar de limbă şi o hartă a cuvintelor.

Lexiconul LEX, menționat anterior, conține un dicționar D_0 alcătuit din cuvinte cu diacritice, un dicționar D_1 ce conține cuvinte fără diacritice care este folosit pentru a genera spațiul de determinare a ipotezelor statistice pentru textul curent. În plus, sistemul mai construiește o listă de cuvinte din textul curent care nu sunt în dicționarele anterioare dar care pot fi considerate erori de tipografie: lexiconul D_2.

- D_0 – este un subset al LEX-ului ce include toate cuvintele care au în alcătuirea lor cel puțin un semn diacritic;
- D_1 – este versiunea fără diacritice a LEX-ului; ar trebui să se păstreze în minte faptul că intrarile în D_0 corespunzătoare cuvintelor-A pot să difere unul de altul doar prin informația POS.
- D_2 – conține cuvinte din textul curent care nu sunt nici în D_0 nici în D_1 care pot fi suspectte că sunt greșeli tipografice: acestea provin din cuvintele $D_0 \cup D_1$ cu diferența de un singur caracter sau prin interschimbarea a două caractere consecutive (în plus, caracterele inverste ar trebui să fie vecine din punct de vedere al poziției pe tastatură.

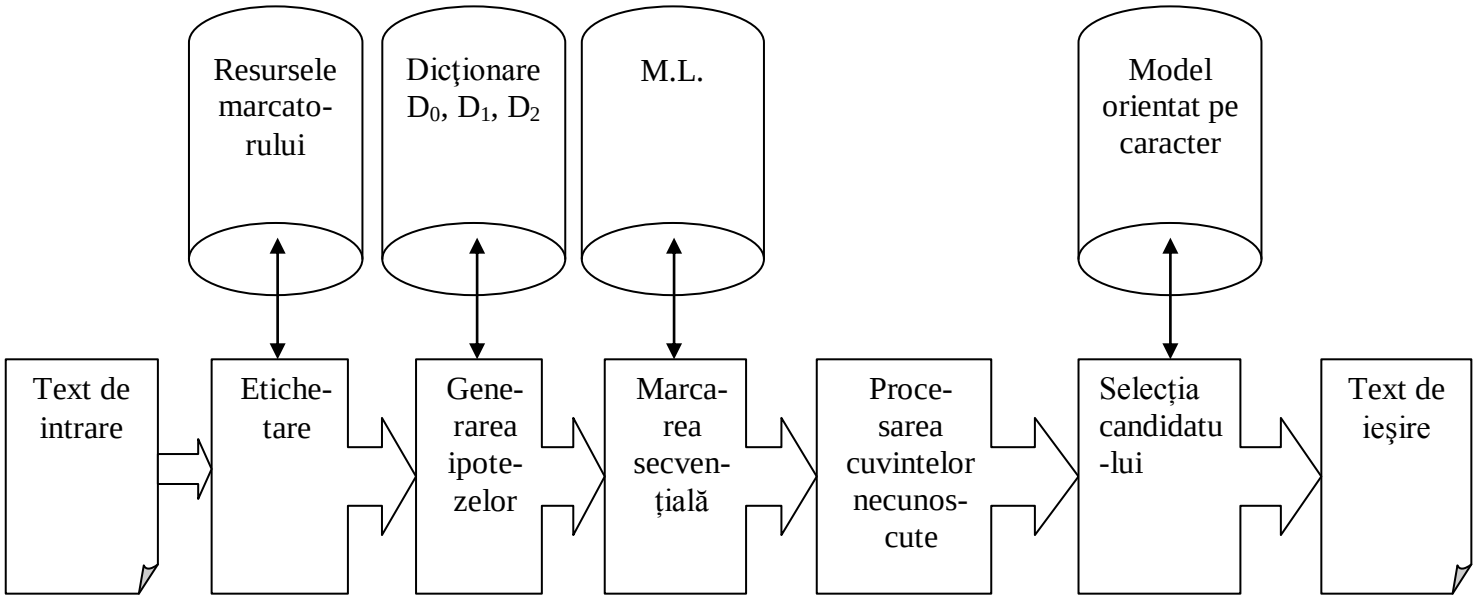

Fig. 1 Arhitectura generala a sitemului DIAC$^+$

Procesul inserției automate a diacriticelor în limba română are patru etape:

i. *Marcarea.* Textul de intrare este segmentat în etichete lexicale conform regulilor specifice resurselor externe.

ii. *Generarea ipotezelor.* În etape generării ipotezelor, fiecare cuvânt este prima dată căutat în reuniunea mulțimilor dintre dicționarelor D_0 și D_1 deoarece în textul fără diacritice sau cu diacriticele parțial inserate nu se poate ști sigur dacă cuvântul este în forma lui regulară sau nu doar dacă informația legată de context este disponibilă.

Dacă cuvântul nu este găsit nici în reuniunea dintre D_0 și D_1, atunci el este căutat în dicționarul D_2. Un cuvânt care nu este găsit în niciun lexicon al sistemului, el este considerat necunoscut și irecuperabil, și procesarea acestuia este lăsată în seama modulului de recuperare la nivel de caracter.

În acest pas, cuvântul W, care apare în textul curent, poate fi asociat cu mai multe intrări din lexiconul LEX și prin urmare pot fi asociate cu un set de perechi <tipar-cuvânt$_k$ MSD$_k$>, cu accepțiunea că versiunea fără diacritice care apare pe tipar-cuvânt$_k$ și din W este identică. Informația furnizată de următoarea secvență va fi folosită pentru filtrarea acestui set și eventual pentru selectarea formei-de-suprafață corecte din punct de vedere contextual.

iii. *Marcarea secvențială.* Textul este marcat secvențial (prin etichete din setul-C redus, apoi fiecărei etichete-C îi corespunde o etichetă MSD de către programul de marcare ME (Ceușu, 2006); pentru această atapă, doar MSD-urile sunt luate în considerare la pasul generării ipotezelor). În cazul existenței cuvintelor necunoscute, marcatorul alege alternativa optimă rezultată din maximizarea entropiei modelului. Pentru marcarea textelor cărora le lipsesc total sau parțial diacriticele se folosesc modele de limbă HMM care reprezintă tranzițiile probabilităților calculate din corpusul de antrenare care conține diacritice, iar probabilitățile propagate sunt calculate din lexiconul fără diacritice. În acest mod, clasele de ambiguitate ale cuvintelor din dicționarul probabilistic și din lexiconul probabilistic POS au fost modificate, dar probabilitățile de tranziție au rămas la fel. De exemplu, cele două cuvinte ambigue: *peste* (Spsa) și *peşte* (Ncms) din corpusul lipsit de diacritice vor fi reprezentate de același tip de etichetă: *peste*, care în acest caz au devenit ambuități de tip POS (Spsa sau Ncms). Este clar că falsa ambiguitate creată de absența diacriticelor scade acuratețea marcatorului, dar aceste erori de etichetare nu sunt vătămătoare pentru procesul de restaurare a diacriticelor.

iv. *Selecția candidaților.* Cuvintele-U sunt înlocuite cu echivalentele lor cu semne diacritice. Un cuvânt-A care nu face parte din setul-S este înlocuit în funcție de *tiparul său* identificat în MSD, atribuit de către marcator, către un anumit cuvânt-A. Pentru cuvintele-S, în funcție de varianta sistemului DIAC$^+$, fie utilizatorul va trebui să aleagă între variantele puse la dispoziție, sau înlocuirea este făcută automat pe baza probabilităților lexicale, sau a anumitor probabilități preferențiale.

v. *Procesare cuvintelor necunoscute.* Această metodă este utilizată ca plan de rezervă în etapa alegerii candidatului ideal, când nicio tipar echivalent al cuvântului nu a fost găsit. Acest caz este destul de rar, deoarece foarte puține cuvinte sunt în afara vocabularului de 800.000 de intrări lexicale. Etapa procesării cuvintelor necunoscute poate fi concepută să funcționeze în paralel cu etapa alegerii candidatului. Pentru procesarea cuvintelor necunoscute s-a folosit un model limbă bazat pe n-grame similar cu cel folosit în (Mihalcea, 2002).

Ordinul modelului	Perplexitatea	Acuratețea (fără spații)	Dimendiunea modelului
2-grame	12,42	93,67%	20,8 KB
3-grame	9,72	95,52%	223 KB
4-grame	7,11	97,72%	1,29 MB
5-grame	5,77	98,59%	4,82 MB
6-grame	5,29	98,84%	13,1 MB
7-grame	5,17	98,84%	27,7 MB
8-grame	5,18	98,85%	48,4 MB

Tab. 8 Evaluarea diverselor modele de limbă orientate pe caracter în procesare cuvintelor necunoscute

S-a optat pentru estimarea Viterbi cu un model de 5-grame pentru a găsi cel mai probabil șir de caractere pentru un cuvânt necunoscut. S-a folosit SRILM – SRI Language Modeling Toolkit (Stolcke, 2002) pentru antrenarea diverselor modele orientate pe caracter. Corpusul de antrenare conține 5.124.277 caractere (inclusiv caracterele de spațiere) în 48.308 propoziții și corpusul de text conține 613.234 caractere în 6.411 propoziții. **Error! Reference source not found.** expune o comparație între perplexitate, acuratețe pentru modele de limbă de diferite ordine.

2.1.5 EVALUAREA

Pentru evaluarea este necesar un corpus de referință R, ce conține aproximativ 118.000 de cuvinte și 502.000 de caractere. Corpusul de referință a fost marcat de mână și analizat. S-au extras toate diacriticele din R dar s-a păstrat marcarea originală. Noua versiune a lui R este ceea ce se numește text marcat ideal pentru DIAC$^+$ (TT): acesta nu are erori de marcare sau de etichetare, și niciun caracter nu deține diacritice. Execuția DIAC$^+$ asupra TT a furnizat o evaluare asupra limitei superioare a acurateței sitemului (când marcarea perfectă este posibilă).

Pentru un scenariu cât mai realist s-au exclus din TT etichetele asociate, rezultând un text etichetat dar neprelucrat (RT) asupra căruia se va aplica lanțul procesărilor (etichetarea cu etichete-C, realizarea corespondenți între această etichetă și eticheta MSD). În ambele experimente DIAC$^+$ a fost utilizat fără intervenția vreunui alt utilizator (procesarea cuvintelor-S se face automat). Rezultatul acestor experimente sunt sintetizate în **Error! Reference source not found.**. Spre deosebire de datele statistice ale Tab. 2, aici nicio etichetă nu a fost exclusă din procesul de evaluare.

Pentru a putea evalua acuratețea inserării diacriticelor s-a dezvoltat un sistem de referință. Sistemul de referință a fost construit folosind corpusul Agenda (10 milioane de etichete). Sistemul folosește un dicționar de tipare ale cuvintelor lipsite de diacritice, împreună cu formele lor corespunzătoare ordonate după numărul aparițiilor. Sistemul de referință înlocuiește formele fără diacritice cu cel mai frecvent tipar de cuvânt. Tiparele corecte de cuvinte diferă în cele două experimente cu un procent de 1,27%. Această diferență poate fi atribuită în totalitate erorilor de etichetare, dar așa cum s-a menționat mai devreme, nu toate erorile de etichetare generează erori ale restaurării diacriticelor. O parte importantă din tiparele incorecte au fost cuvinte-S (321), care ar fi trebuit să primească atenția utilizatorului, care va trebui să-și exprime opțiunea sa.

În Tab. 5 se pot observa rezultatele de evaluare ale acelorași experimente, dar de data aceasta procesarea textului se face la nivel de caracter. Așa cum s-a putut observa, acuratețea evaluării la

nivel de caracter arată o performanță de peste 99% chiar și pentru sistemul de referință. Este nevoie să se țină cont că evaluarea la nivel de caracter (0,6%) arată mult mai bine decât eroarea la nivel de cuvânt (2,25%). Aceasta confirmă teoria inițială că se poate estima cu ușurință rezultatul evaluării (la nivel de cuvânt sau la nivel de caracter) cunoscând celălalt rezultat, iar dacă lungimea media a cuvintelor este mai mică decât distanța medie dintre două caractere cu diacritice.

Cu toate acestea, pe baza fișierului de înregistrare, aceste erori pot fi cu ușurință corectate. Restul tiparelor incorecte rezultă din erorile de marcare. Unele dintre aceste erori de marcare, în limba română, sunt foarte dificil de rezolvat dat fiind un context limitat.

	DIAC cu text etichetat	**DIAC cu text neprelucrat**	**Sistem de referință**
Etichete	117.909		
Cuvinte fără diacritice	34.745 (29,47%)		
Cuvinte-S	361		
Cuvinte necunoscute	2.130 (1,8%)		
Tipare corecte	116.810 (99,06%)	115.262 (97.75%)	113,491 (96,25%)
Tipare incorecte	1.092 (0,94%)	2.609 (2,25%)	4.418 (3,75%)

Tab. 9 Evaluarea acurateței tiparelor în DIAC$^+$

Cele mai multe erori provin din atributul care se referă la timpul verbelor (prezent sau imperfect) la conjugarea I, forma la modul infinitiv care se termină cu „a”. Rezolvarea acestei erori necesită o post-etichetare care implică o inspecție a propozițiilor vecine, precum și o analiză timpurilor unor secvențe de cuvinte (în speranța că verbele din vecinătate nu sunt în aceeași clasă de conjugare).

	DIAC cu text etichetat	**DIAC cu text neprelucrat**	**Sistem de referință**
Caractere (fără spații)	501.735		
Semne diacritice	41.144		
Caracter corecte	500.400 (99,73%)	498.764 (99.40%)	497.096 (99,07%)
Caractere incorecte	1.335 (0,27%)	2.971 (0,6%)	4.639 (0,93%)

Tab. 10 Evaluarea acurateței caracterelor în DIAC$^+$

2.2 METODA RESTAURĂRII LA NIVEL DE CARACTER

Metoda este propusă de Rada F. Mihalcea și Vivi A. Nastase. În lucrarea lor „**O metoda automata pentru inserarea diacriticelor în texte în limba română**” s-a descris o metoda de restaurare a diacriticelor bazată pe tehnici de învațare la nivelul de literă. Avantajul principal al metodei consta în capacitatea ei de generalizare dincolo de cuvinte. Nu este necesara nici un fel de analiza a textului, si nu se folosesc nici un fel de procesoare de limbaj sau dictionare. Singura cerință este un corpus relativ mic de texte cu diacritice.

Metoda este folositoare în special pentru limbi pentru care nu sunt disponibile dicţionare electronice de dimensiune adecvate, şi nici procesoare pentru analiză morfologică şi/sau sintactică. Mecanismul de învăţare foloseşte date de intrare extrase din texte neprelucrate, şi

generează rezultate cu o precizie ridicată. Experimente detaliate efectuate pe texte în limba română au arătat că restaurarea diacriticelor în această limbă se poate efectua folosind metoda propusă cu o precizie de peste 99% la nivel de literă.

Rezultatele au fost validate prin experimente efectuate pe alte trei limbi europene care fac uz de diacritice: cehă, poloneză şi maghiară. Precizie medie măsurată pe cele patru limbi de studiu este de 98.14%, fapt care demonstrează că metoda este independentă de limbă. În plus, un alt avantaj al metodei este faptul că, datorită simplităţii sale, viteza de procesare este foarte mare, de până la 20 pagini de text pe secundă.

2.2.1 DESCRIERE GENERALĂ

Lucrarea de față prezintă o metodă de restaurare a diacriticelor bazată pe învățarea la nivel de literă, și nu la nivel de cuvânt. Avantajul principal al acestei metode este faptul că oferă posibilitatea de generalizare dincolo de cuvinte. Metodă este folositoare mai ales pentru limbile pentru care resursele disponibile sunt limitate, în speță limbi care nu au dicționare electronice mari cu diacritice. Limbi cunoscute și bine studiate, precum franceză și spaniolă, pot de asemenea beneficia de această metodă pentru procesarea cuvintelor necunoscute.

Experimentele prezentate în această lucrare adresează în principal problema restaurării diacriticelor în texte din limbă română. Precizia observată pentru limbă română este de 99%, măsurată la nivel de literă. Experimente similare au fost efectuate pe alte trei limbi, și anume poloneză, maghiară și cehă, de asemenea cu rezultate foarte bune.

Avantajul principal al metodei este faptul că nu necesită nici o etapă de preprocesare, ci numai un corpus relativ mic format din texte cu diacritice. Datorită simplității algoritmului, viteză de procesare este foarte mare, de aproximativ 20 pagini de text pe secundă, măsurată pe un calculator cu un procesor Pentium III cu frecvență de 500MHz și 250MB memorie.

Practic, metodă propusă încearcă să învete reguli aplicabile la nivel de literă. În loc de a învăța reguli care se aplică la nivel de cuvânt, cum ar fi „anuncio se scrie anunció atunci când are funcția de verb”, se dorește a învăța reguli aplicabile la nivel de literă, cum ar fi „s urmat de i și spațiu și precedat de spațiu se scrie s”. Astfel de reguli, învățate la nivel de literă, sunt mai generale și au aplicabilitate mai mare, în special în cazurile în care dicționarele disponibile sunt de dimensiune redusă, când se întâlnesc multe cuvinte necunoscute în textul dat, sau când procesoare pentru analiză morfologică sau sintactică nu sunt la îndemâna.

Este evident că în analiză limbajului literele constituie nivelul cu granularitatea cea mai scăzută, și de aceea au și cel mai mare potențial de generalizare. În loc de aproximativ 150.000 de unități candidate potențiale pentru algoritm (mărimea aproximativă a vocabularului de uz general a unei limbi), vom avea mai mult sau mai puțin 26 caractere pe bază cărora se vor constitui datele de intrare pentru algoritmul de dezambiguare.

EXPERIMENTE

Scopul experimentelor descrise în această lucrare este de a arată că învațarea la nivel de literă este posibilă și poate rezolvă, cu precizie mare, problemă restaurării diacriticelor. Pe lângă faptul că metoda propusă constituie o problemă de cercetare, scopul învățării la un nivel de granularitate atât de scăzut este de a oferi o metodă viabilă pentru limbile pentru care resursele lexicale și semantice disponibile sunt limitate, și pentru care restaurarea diacriticelor prin învatare la nivel de cuvânt este greu de realizat.

DATE

Se vor prezenta experimentele efectuate pe texte în limba română. Limba româna nu este, aşa cum s-a mai menţionat în acest capitol, o limba foarte raspândită, şi în consecinţă nu are foarte multe resurse public disponibile pentru pre-procesare, iar dicţionarele electronice sunt de dimensiuni relativ mici.

Pentru a aplica metoda descrisă în lucrarea de fata, este nevoie de o colectie de texte românesti cu diacritice. În acest scop, s-au colectat articole din „România Literara", un ziar românesc publicat săptamânal, cu articole legate în general de literatură.

Ziarul are o versiune care contine diacritice începând din anul 2000. Colectia disponibilă on-line la data colectării datelor (august 2001) cuprindea 2780 articole. În pasul urmator, textul a fost extras din fişierele HTML. Atenţie deosebită a fost acordată doar caracterelor românesti. Alte caractere cu diacritice întâlnite ocazional, cum ar fi c, é, etc. au fost transformate în forma lor echivalentă, fără diacritice. Dupa toate aceste faze premergătoare, s- obtinut un corpus conţinând aproximativ 3 milioane de cuvinte.

Literele mari au fost transformate în litere mici. Cazul literelor â si î este special în limba română: deşi pronunţia lor este identică, folosirea lor este guvernată de reguli bazate pe pozitia lor în cuvânt. La începutul cuvântului se foloseşte întotdeauna î, iar â se foloseşte în interiorul cuvântului. Este bine cunoscut faptul că folosirea acestor litere a fost controversată de-a lungul timpului. O lege din anii `60 a schimbat ortografierea de la â la î, singura excepţie fiind cuvintele derivate din radacina român. La începutul anilor `90 ortografia veche a fost reintrodusă, şi astfel s-a ajuns la cazuri de texte inconsistente, în care se întâlnesc scrieri diferite ale aceluiasi cuvânt. De exemplu, cîntec si cântec sunt forme ale aceluiasi cuvânt care pot fi întâlnite în acelasi text. Ziarul „România Literară" păstrează înca ortografia cu î, cu mici exceptii (de exemplu, articole scrise de scriitori invitati care preferă să scrie folosind â în loc de î).

2.2.2 ALGORITMI DE ÎNVĂŢARE

Pentru a rezolva problema restaurării diacriticelor, s-a ales folosirea unui algoritm bazat pe învăţarea de instante (IBL). Există două motive importante care au stat la baza luării acestei decizii. În primul rând, este un fapt demonstrat că excepţiile au un rol important în procesarea limbajelor naturale. Algoritmii de tip IBL sunt recunoscuţi pentru faptul ca iau considerare fiecare exemplu de antrenament în luarea unei decizii de clasificare, deci folosirea acestui tip de algoritmi prezintă un avantaj deosebit în problemele de limbaj natural. În al doilea rând, acest gen de algoritmi sunt foarte eficienţi relativ la timpul de antrenament şi testare.

Învăţarea pe baza de instante se desfăşoara în felul urmator: în pasul de antrenament, toate exemplele de intrare sunt memorate. În faza de testare, fiecare exemplu din set este comparat cu exemplele memorate, şi va primi clasificarea dată de exemplul memorat de care este cel mai apropiat, distanţa fiind dată de măsura specifică, aleasă în implementarea folosită.

Pentru efectuarea experimentelor propuse, s-a folosit implementarea TiMBL a acestor algoritmi. În plus, au fost efectuate experimente asemănătoare şi cu un clasificator pe bază de arbori de decizie, si anume C4.5. Arborii de decizie sunt construiţi din setul de exemple de antrenament. La fiecare pas este ales un atribut care discriminează cel mai bine exemple din clase diferite (prin valorile sale). Grupele obţinute prin diviziunea după acest atribut vor fi din nou împărţite în grupe mai mici şi mai pure, prin alegerea unui nou atribut care discriminează cel mai bine exemplele din grupa. Acest proces continuă până când grupele obtinute au un grad de puritate acceptabil, sau mărimea arborelui depăşeşte un prag ales iniţial. Rezultatele obţinute folosind

C4.5 sunt asemanatoare cu cele obținute folosind TiMBL, însă C4.5 are capacitatea de a genera reguli expresive, folositoare pentru implementări practice.

Având în vedere că se lucrează la nivelul literelor, atributul care trebuie învătat este constituit de litera ambiguă. Acesta poate fi oricare din literele ambigue enumerate în Tabelul 1. Pentru limba româna avem 4 perechi de litere ambigue: s - ș, t - ț, a - â, i - î. Literele mari au fost convertite în prealabil în litere mici. Datorita faptului că datele folosite aplică ortografia cu î, nu avem ambiguitatea a - â, ci doar ambiguitatea i - î. Aceasta nu implica însă o pierdere de generalitate. Conversia între cele doua forme de ortografie este simplă și se poate realiza folosind doar poziția literei în cuvânt și, prin urmare, scrierile diferite nu afectează rezultatul algoritmului.

ATRIBUTE

Atributele folosite în orice algoritm de învatare au un impact foarte mare asupra eficacității algoritmului. Dupa cum s-a mentionat și în introducere, nu există posibilitatea folosirii procesoarelor care determină partea de vorbire a cuvintelor, și nici un alt fel de analizoare morfologice sau sintactice. În plus, nu se face nicio corelație între cuvintele învecinate, deoarece există un număr limitat de date, și în consecință exista șansa de a întâlni un număr mare de cuvinte necunoscute. Prin urmare, se folosesc niște atribute foarte simple, pentru extragerea cărora nu este nevoie de nici un fel procesare specială. Se vor folosi litere învecinate, cu o notație specială atribuită spatiilor, virgulelor și punctelor (aceste caractere pot afecta procesul de învatare, fiind considerate caractere speciale de către C4.5 și/sau TiMBL).

Daca X este litera a carui ambiguitate trebuie rezolvată, atributele folosite sunt N litere la stânga și la dreapta literei ambigue:

$$L_{-N}, L_{-(N-1)}, \dots, L_{-1}, X, L_1, L_2, \dots, L_{(N-1)}, L_N$$

Acest set de atribute se comportă surprinzator de bine, relativ la acuratețe, după cum se va arăta în cele ce urmează. După cum s-a mentionat mai devreme, procesul de restaurare nu se bazează pe nici un tag obținut cu procesoare lexicale sau morfologice, ci doar pe informația care se poate extrage din textul neprelucrat. De asemenea, s-au căutat posibilități de generalizare, astfel încât un corpus limitat să poată fi folosit pentru a genera reguli de reinserare a diacriticelor. În loc de a învața reguli pe baza pe cuvinte, după cum s-a procedat pâna acum, se dorește învățarea unor reguli bazate pe litere, pentru că acestea constituie cele mai mici unitati în limbaj, și oferă posibilitatea învățării chiar și dintr-o colectie mică de texte. Pentru fiecare pereche ambiguă de litere, se parcurge textul și se generează toate exemplele posibile întâlnite în text. Atributele, de exemplu, sunt formate folosind N litere la stânga și la dreapta literei ambigue, și atributul țintă este însăși litera ambiguă. Forma generală a exemplelor generate este:

$$L_{-N}, L_{-(N-1)}, \dots, L_{-1}, L_1, L_2, \dots, L_{(N-1)}, L_N, X$$

unde ca și în exemplul anterior, X este litera ambiguă. Se pot prezenta mai jos exemple de vectori de atribute care constituie date de intrare pentru algoritmul de învatare pentru rezolvarea ambiguității perechii s - ș. CO, DO si SP sunt codurile care înlocuiesc virgula, punctul și spațiul.

l , i , n , SP , (, u , b , SP , i , n , s.

e , CO , SP, r , o , - , g , a , r , d, ş.

g , a , r , d , i , t , u , l , CO , SP , s.

e , SP , o , r , a , DO , SP , t , o , t , ş.

Învățarea se reduce la detectarea corelațiilor între valorile atributelor care caracterizează exemplele de antrenament și valorile atributelor țintă, și utilizarea acestora pentru stabilirea valorii atributului țintă din exemplele de testare.

Numarul de exemple extrase din corpus depinde de perechea de litere. Din întregul set de 3 milioane de cuvinte, s-au obținut 2.161.556 exemple pentru perechea ambiguă a - ă, 2.055.147 pentru perechea i - î, 1.257.458 exemple pentru t - ț, și în final 866.964 exemple pentru perechea s - ș. În fiecare din aceste cazuri, spațiul exemplelor este împărțit în două clase, date de cele 2 variante ale literei ambigue. Metoda de învățare automată va folosi atributele date pentru a găsi reguli de clasificare a exemplelor în cele 2 clase.

2.2.3 REZULTATE

Precizia cea mai ridicată s-a obținut pentru o fereastra de 10 litere în vecinătatea literei ambigue (N = 5). Dată fiind această observație, s-a considerat că este important să studieze mai in detaliu acest caz, și să se determine ratele de învățare pentru cele 4 perechi de litere ambigue. Cu toate acestea, s-au obținut rezultate pentru ferestre de diverse dimensiuni, pentru comparație. Tabelul 2 arată rezultatele obținute pentru N=5. Autorii sistemului au condus experimente cu seturi de antrenament de diverse dimensiuni, variind de la 2.000.000 exemple pâna la 10 exemple, pentru a determina rata de învățare și dimensiunea minimă a corpusului necesară pentru a obține o precizie satisfacătoare. În toate aceste experimente s-au folosit seturi de testare continând 50.000 exemple. Pentru a obține rezultate cât mai precise s-a folosit validare încrucișată folosind 10 seturi diferite de test.

Este interesant de observat ca cea mai importanta fază a procesului de învatare are loc când se folosesc primele 10.000 exemple. În conformitate cu măsuratorile efectuate, a rezultat ca aproximativ 100.000 – 250.000 caractere (aproximativ 25-60 pagini de text) sunt necesare pentru a genera 10.000 exemple cu diacritice, ceea ce constituie un corpus de dimensiune relativ mic. Mai departe, pentru a obtine îmbunătățiri de numai 1% este necesar un număr semnificativ de exemple însă și numai 1.000 exemple sunt suficiente pentru învățare.

	Perechea ambiguă				
	a-ă	a-ă (2)	i-î	s-ș	t-ț
Nr. total exemple	2.161.566	1.369.517	2.055.147	866.964	1.157.458
Baza comparație	74,70%	85,90%	88,205%	76,53%	85,81%
Exemple de antrenament	Precizia obținută pe date de test (50.000 exemple)				
2.000.000	96,14%	99,14%	99,69%	99,07%	98,75%
500.000	94,57%	98,79%	99,46%	98,86%	98,40%
2000	85,81%	89,93%	95,49%	93,47%	93,56%
100	74,80%	84,04%	91,41%	82,13%	84,46%
10	73,38%	85,90%	88,20%	75,88%	85,81%

Tab. 11 Rezultate obținute în rezolvarea ambiguității literelor cu dicritice în limba română

Folosind întregul set de exemple extrase din corpus, rezolvarea ambiguităţii perechii i - î este aproape 100% corectă. Pentru aceasta diacritică, avem acum o instanţă greşită din 300 instante, în timp ce bază de comparaţie implica o instanţă greşită din fiecare 8 instante, deci o îmbunătăţire semnificativă.

Cel mai slab rezultat este obţinut în cazul perechii a - ă. După o analiză a rezultatelor, reiese că principalul motiv care cauzează această precizie scăzută este faptul că multe substantive în limba româna au formă nearticulată terminată în ă şi forma articulată terminată în a. De exemplu, „masă” şi „masa” reprezintă forma articulată şi respectiv nearticulată a substantivului masa. De asemenea, timpuri diferite ale aceluiaşi verb se disting numai prin terminaţia în a sau ă. Algoritmul de învăţare este deci indus în eroare din cauza folosirii acestor litere în contexte identice. O soluţie simplă constă în evitarea în procesul de învăţare a exemplelor care conţin a

sau ă la sfârşitul unui cuvânt. Rezultatele obţinute sub această ipoteză simplificatoare sunt raportate în Tab. 11 Rezultate obținute în rezolvarea ambiguității literelor cu dicritice în limba română, în coloana a-ă (2). După cum se arată în tabel, câştigul este de mai mult de 4% în precizie folosind doar această condiţie simplă (câştig care se traduce într-o reducere a erorii de 87%).

S-a folosit de asemenea şi algoritmul de învăţare bazat pe arbori de decizie C4.5, cu aceleaşi date de antrenament, fără a observa însa nici o îmbunătăţire comparativ cu rezultatele raportate în Tab. 11 Rezultate obținute în rezolvarea ambiguității literelor cu dicritice în limba română. Dezavantajul folosirii C4.5 pentru această problemă este faptul că faza de învăţare este mult mai lentă decât în cazul folosirii algoritmului TiMBL.

Pe de altă parte, C4.5 are capacitatea de a genera reguli expresive. „Dacă L_1=e şi L_2=spaţiu atunci ș”(99.5%), „Dacă L_1=t şi L_2=spaţiu atunci ș” (98.7%), „Dacă L_{-4}=p şi L_{-1}=v şi L_1=t şi L_2=e atunci ș”(95.5%), sunt exemple de astfel de reguli. L_i denotă o literă învecinată în poziţia i relativ la litera ambiguă. Se observă că aceste reguli nu ţin cont de faptul că literele folosite în clasificare aparţin aceluiaşi cuvânt sau nu. Algoritmul de învăţare se bazează pur şi simplu pe litere, indiferent de cuvântul căruia îi aparţin. În consecinţă, pseudo-omonimele (cum ar fi peste şi peşte), sunt adresate în mod egal de această metodă, pentru că algoritmul are capacitatea de a se extinde dincolo de cuvinte.

2.2.4 COMPARAȚIE CU EXPERIMENTE ASEMĂNĂTOARE

Rezultatele prezentate în lucrarea de față se pot compara cu rezultatele raportate de Tufiş şi Chiţu, care au folosit tot limba româna în experimentele lor. Tufiş şi Chitu menţionează că sarcina recuperării diacriticelor în limba româna este mai dificilă decât în alte limbi, deoarece în română diacriticele sunt mai intens folosite. După cum raportează în experimentele lor, numai 60% din cuvintele din limba româna nu au diacritice, comparat cu studii menţionate în care arată că aproximativ 85% dintre cuvintele limbii franceze se scriu fără accent.

Abordarea prezentată de Tufiş şi Chitu foloseşte dicţionare, un analizor morfologic, iar învăţarea se face la nivel de cuvinte. Folosind aceste resurse, au obţinut o precizie globală de 97.4%. Nu putem efectua o comparaţie directă a rezultatelor noastre, având în vedere că atât metodele, cât şi modul de evaluare, sunt fundamental diferite.

Precizia medie de 99% pe care noi o raportăm este măsurată la nivel de literă, pe când acurateţea raportată în este determinată la nivel de cuvânt.

Metodologia noastră depăşeşte abordările anterioare, prin faptul că s-au obţinut precizii şi viteze de procesare ridicate fără a folosi nici un fel de resurse adiţionale cum ar fi procesoare pentru analiza morfologica sau sintactica sau dicţionare. Din aceste motive, algoritmul se poate aplica oricărei limbi, singura cerinţa fiind un corpus relativ mic de texte cu diacritice.

2.3 ALTE TIPURI DE MODELE DE LIMBAJ

În afara modelelor de limbă formate din n-grame reprezintă stadiul actual al tehnicii de modelare de limbaj.

O gramatică bazată pe bucle ale cuvintelor este un model care atribuie probabilităţi egale tuturor cuvintelor din vacabular şi, implicit, tuturor secvenţelor de cuvinte. Acest tip de model de limbă ar putea fi, de fapt, un model de limbă format din unigrame cu probabilităţi egale ale unigramelor (nu estimate din corpusul de formare). O gramatică bazată pe bucle ale cuvintelor oferă rezultate bune pentru recunoaşterea cuvintelor izolate.

O gramatică cu stări finite sau o gramatică în reţea este un model bazat pe principiul grafurilor în care nodurile sunt cuvintele iar legăturile reprezintă posibilele tranziţiiale cuvintelor. O gramatică cu stări finite specifică explicit toate secvenţele posibile de cuvinte pentru un domeniu ales. Costuri specifice pot fi atribuite nodurilor existente, specificând, totuşi, diferitele probabilităţi pentru toate secvenţele de cuvinte acceptate. Dacă sarcina modelului de limbă nu este una complicată (recunoaşterea cifrelor, a numerelor de telefon, selecţia dintr-un meniu) atunci acest model de limbaj poate fi folosit cu succes. Mai mult de atât, gramatica cu stări finite poate fi folosită cu succes pentru aplicaţiile care găsesc cuvinte potrivite.

CAPITOLUL 3

TEHNOLOGII PROCESARE

„Pentru orice realizare primul pas este curajul”

Johann Wolfgang van Goethe

3.1 JAVANLP – PROCESAREA TEXTULUI

NLP este un domeniu al informaticii, inteligenței artificiale, și lingvisticii computaționale care se ocupă cu interacțiunile dintre mașină (calculator) și vorbirea umană.

Acest domeniu se ocupă cu precădere de: clasificări de text, detecția plagiatului, generarea vorbirii, de realizarea traducerilor automate, a recunoașterii vorbirii, a sumarizării automate, a analizei de sentimente și, nu în ultimul rând, a corecției gramaticale.

Pentru realizarea procesării limbajului trebuie introdus conceptul de învățare computațională (Machine Learning – ML). ML se bazează pe principiul că lumea reală funcționează după o anumită regularitate, astfel că, regulile pot fi exploatate pentru a îmbunătăți rezolvările aspectelor cu care se confruntă omul/mașina. Arthur Samuel comenteză despre acest domeniu că este „un domeniu de studiu care oferă mașinilor (computerelor) abilitatea de a învăța fără să fie explicit programate” (1959), iar Tom Mitchell afirmă că „un program de calculator poate învăța din experiența E, prin rezolvarea unor sarcini S cu eficiența măsurată P, dacă eficiența rezolvării sarcinii S, măsurată de către P, crește cu experiența E” (1997).

Petru a putea realiza procesul de restaurare a diacriticelor asupra oricărui text, chiar și pentru un text în format HTML, acesta trebuie mai întâi să fie transformat în text simplu, și să fie reținute doar datele care conțin un discurs coerent pentru limba română. Spre exmplu, o imagine într-un fișier HTML care prezintă un articol de ziar, ar putea crea probleme sistemului. În afara textului cu pricina, fișierul mai poate conține multe taguri HTML, URL-uri, elemente ce faceau parte din meniuri și multe alte reclame. Toate aceste părți trebuie să fie înlăturate, deoarece interesul se manifestă doar asupra textului în cauză.

Un utilitar NLP creat în scopul recondiționării corpusului de text este aplicația de curățare a textului javaNLP implementat de Horia Cucu în 2011. Acest utilitar este inclus în proiectul prezentat în proiectul de față. Acestă aplicație este scrisă în limbajul de programare Java și, în consecință, poate fi executată pe orice sistem de operare care are instalat Mașina Virtuală Java (JVM). Operația de curățare a textului preia un corpus ca fișier de intrare și aplică, pe rând, mai multe operații de procesare cu scopul final de a obține un text simplu, fără cifre, semne de punctuație sau caractere speciale.

Prima operație de curățare are rolul de a înlocui toate caracterele cu diacritice cu un caracter unic per semn diacritic. În corpusuri diferite sau chiar într-o altă parte a aceluiași text, caractere Unicode distincte sunt folosite pentru a exprima același înțeles (ş” cu sedilă vs. “ș” cu virgulă, “ţ” sedilă vs. “ț” cu virgulă, etc.). Această inconsecvență este rezolvată de prima operație de curățare.

În multe articole de ziare și în special în textele ce aparțin Parlamentului European sunt folosite multe abrevieri. De exemplu *dnă*, *dnă.* și *d-nă ,* toate înseamnă „doamnă”, iar *dl* și *dl.* semnifică *domnul*, *art.înseamnă articolul,* etc. Nu se poate realiza un model de limbă care prezice abrevierile și care să recunoască formele lor fără abrevieri. Mai mult, multe forme abreviate pentru același cuvânt din datele de antrenare produc inconsistențe în estimarea probabilităților pentru un cuvânt particular. Cea de-a doua operație de curățare înlocuiește abrevierile cu formele complete ale cuvintelor dintr-o listă de abrevieri. În acest punct, doar o listă de 30 de intrări a fost creată și folosită, dar operația de curățare va deveni mai eficientă pe măsură ce lista se va extinde.

Numerele scrise cu cifre în loc de litere reprezintă o problemă similară, la fel ca și abrevierile. În afară de numerele simple, corpusurile mai pot conține și alte numere cu diverse înțelesuri sau formate. *01.02.2004* (date calendaristice), *al 30-lea* (numeral ordinal), *2.59%* (numere fracționale), *3 000 000, 3.000.000, 3,000,000* (numele mari în diferite formate), 10:20, 10h20, 10.20 (timpul reprezentat și el în formate diferite), etc. Doar cele mai frecvente formate au fost transformate în text.

Cea de-a patra operație de curățare se ocupă de semnele punctuație și de alte caractere speciale. Astfel, punctele, virgulele, semnele de întrebare și de exclamare au fost înlocuite cu un carater de linie nouă (în acest fel va apărea o singură propoziție pe linei în fișierul de ieșire), iar parantezele au fost și ele îndepărtate, textul dintre ele a fost trecut pe o linie nouă. Cratima este pur și simplu îndepărtată cu excepția construcțiilor (într-o sau dă-mi-le), iar caracterul „%” este înlocuit cu expresia „la sută”.

Cea de-a cincea operație de curățare îndepărtează toate liniile (o linie reprezintă de fapt o propoziție) care au mai puțin de trei cuvinte. Aceste propoziții foarte scurte sunt de cele mai multe ori și cele cu un efect defectuos ridicat, deoarece acestea sunt abrevieri sau numere pe care operația precedentă nu le-a putut procesa adecvat.

Ultima și cea de-a șasea operație utilizează o listă de propoziții pentru a șterge acele propoziții care au procentul OOV mai mare de 30%. Acest procent a fost ales empiric pentru a echilibra propozițiile care sunt în limba română de cele care sunt scrise într-o limbă străină.

3.2 SRI-LM

Disambig face parte din pachetul SRI-LM. SRI-LM este o colecție de librării în C++, programe executabile și scripturi capabile să permită atât producția cât și experimentarea cu ajutorul modelelor statistice a recunoașterii vocale și a altor aplicații înrudite. SRI-LM este disponibil gratis pentru scopuri necomerciale. Utilitarul suportă creația și evoluția unei mari varietăți de modele de limbă bazate pe N-grame, cât și a unor sarcini înrudite cum ar fi etichetarea statistică și manipularea listelor cu N-grame și mulțimilor de cuvinte.

Așa cum am menționat în capitolele anterioare modelarea statistică de limbaj este știința și mai ales arta de a construi medele care să estimeze probabilitățile unor succesiuni de litere. Modelarea limbajului are multe aplicații în tehnologiile legate de limbajul natural. La momentul actual, pricipalele tehnici pentru o modelare efectivă de limbaj sunt cunoscute de mai bine de un deceniu, deși se dorește o îmbunătățire a aplicațiilor, noi descoperiri propun realizarea acestor modele prin rețele neurale.

Câteva pachete sofware destinate modelării limbajului sunt în uz de mult timp, algoritmii de bază sunt atât de simpli încât cineva le-ar putea implementa ușor cu un efort rezonabil pentru o sarcină de cercetare. În comparație cu alte utilitare LM, SRILM oferă o interfață de programare și un set extensibil de instrumente, multe tipuri care nu fac parte din standard, și o funcționalitate suplimentară care trece dincolo de modelarea de limbaj, incluzând etichetarea și atribuirea diverselor scoruri.

Proiectul a fost influiențat de alte aplicații software înrudite cum ar fi CMU-Cambridge. Prima implementare a fost pentru crearea modelelor bazate pe n-grame în 1995. Principalul scop al serviciului SRI-LM este modelarea și evaluarea limbajului natural. Estimarea înseamnă creația unui model din datele de antrenare; evaluarea înseamnă calculul probabilităților unui corpus de test, exprimat, de obicei, printr-un set de test al perplexității datelor. De vreme ce majoritatea LM sunt bazate pe statistici ale N-gramelor, instumentele necesare pentru a realiza aceste două operații sunt numite *ngram-count* și respectiv *ngram*. Un model de limbă standard (format din trigrame Good-Turing) va fi creat prin următoarea comandă:

```
ngram-count -text TRAINDATA -lm LM
```

LM-ul rezultat poate fi evaluat pe un corpus de test prin:

```
ngram -lm LM -ppl TESTDATA -debug 2
```

„`ngram -debug`" este o opțiune care controlează finețea evaluării. Un nivel 2 înseamnă că probabilitățile vor fi raportate la nivel de cuvânt, incluzând ordinul N-gramelor folosite, pe lângă probabilitățile logaritmice și perplexitatea. Anumite statistici suplimentare care ilustrează calitatea modelui este numărul de cuvinte din afara vocabularului (OOV) și „*rata de potrivire*" pentru diverse nivele de N-grame. SRILM nu realizează de la sine vreo logică asupra cuvintelor și tratează tot ce este între spații ca un cuvânt. Normalizarea și etichetarea textului sunt foarte dependete de corpus și sunt obținute de către filtre care preprocesează datele.

Programele *ngram-count* și *ngram* au un număr vast de opțiuni pentru un control al multor parametrii ale estimărilor LM și testării. Cei mai importanți parametrii pentru antrenarea LM sunt:

- Ordinul N-gramelor. Nu există o limită în contruirea n-gramelor;
- Metoda de reducere folosită (Good-Turing, Witten-Bell sau Kneser-Ney modificată);
- Un vocabular opțional predefinit pentru a extinde setul cuvintelor din datele de antrenare;
- Opțiunea de a lăsa nemodificate cuvintele necunoscute sau de a le înlocui cu o etichetă specială;

- Opțiunea de a ține cont sau nu distincția între caracterele mici și mari.

Dincolo de estimarea modelului, *ngram-count* mai realizează și o manipulare a numărului N-gramelor, cum ar fi generarea numărului de apariții pentru un text, sumarizarea fișierelor, recalcularea numărului de apariții pentru n-grame de ordin mai mic.

Un model de limbă interpolat este obținut prin realizarea unei uniuni a N-gramelor unui model de intrare, atribuind fiecărei N-grame o medie ponderată a probabilităților acelor modele (în unele modele aceste probabilități se pot calcula prin algoritmul de întoarcere) și să se renormalizeze noul model. S-a demostrat că printr-o astfel de tehnică de interpolare se ajunge la o perplexitate mai redusă.

Pe lângă N-gramele bazate metode de întoarcere, SRILM implementează multe alte tipuri de modele de limbă.

Modelele bazate pe clase – N-gramele formate din clase de cuvinte constituie o modalitate eficientă pentru a mări robustețea LM-urilor și să încorporeze cunoștințe legate de domeniu, definind clase de cuvinte cu semnificații diferite dependente de domeniu. SRILM permite membrilor clasei să fie alcătuiți din mai multe cuvinte (de exemplu „Alba Iulia" este un membru al clasei „NUME-ORAȘE"). Acest aspect împreună cu faptul că unele cuvinte pot aparține mai multor clase, necesită utilizarea programării dinamice pentru a evalua clasa de N-grame. Clasele de cuvinte pot fi definite manual sau printr-un program separat prin algoritmul Brown.

Modelele păstrate în memoria volatilă – Această bine cunoscută tehnică LM atribuie probabilități nenule cuvintelor.

Modelele bazate pe evenimente ascunse – Modelele bazate pe acest tip de evenimente ascunse încorporează cuvinte speciale care apare în model, dar nu apar în fluxul de date observat. În schimb, aceste modele corespund stărilor modelui Markov ascuns, și pot fi folosite pentru a modela evenimente lingvistice cum ar fi propoziții care nu sunt limitate. Opțional, aceste evenimente pot fi asociate probabilităților nonlexicale care condiționează LM sau alte surse precum prozodia. Un tip special de eveniment ascuns dintr-un model de limbă poate să abordeze neconcordnțele de limbaj prin faptul că se permite evenimentelor ascuse să modifice istoria cuvintelor; de exemplu, un eveniment de ștergere a unui cuvânt ar elimina unul sau mai multe cuvinte pentru a modela un start fals.

Modelele de limbă cu salt – În acest LM, cuvintele din istorie sunt sărite în mod probabilistic, permițând cuvintelor care sunt la o distanță mai mare pentru să le ia locul. Probabilitățile sărite asociate fiecărui cuvânt sunt estimate prin maximizarea probabilităților.

HMM-urile N-gramelor – Acest model de limbă constă în modelul Markov ascuns (HMM) în care fiecare stare este asociată cu propria distribuție a N-gramelor. Acest model pornește dintr-o anumită stare până când N-gramele ating sfârșitul propoziției, la care punct realizează o tranziție probabilistică către stările vecine. HMM-urile furnizează un cadru de dezvoltare general care poate să codeze o varietare de tipuri de LM propuse în literatură, cum ar fi amestecul la nivel de propoziție.

LM-urile interpolate dinamic – Două sau mai multe LM-uri pot fi liniar interpolate la nivel de cuvânt.

Librăriile de clase C++ implementează API-ul servicului SRILM. Programarea orientată pe obiecte se pare a fi o potrivire excelentă pentru implementarea modelelor de limbă, din cauza mai multor motive. O ierarhie a claselor reflectă ierarhic relațiile de specializare dintre diferite tipuri de LM. Moșteniea permite noilor variante de modele de limbă să poată deriva din cele existente cu un efort minim. Un nou model minimal de limbă are nevoie de definirea doar a unei funcții *wordProb,* o metodă utilizată pentru calculul probabilităților condiționale fiind dat un cuvânt și istoria cuvintelor precedente. Cele mai multe funcții ale unui LM sunt definite generic,

și trebuiesc reimplementate pentru o clasă nouă de LM. De exemplu, *sentenceProb* este definit prin intermediul *wordProb* și moștenește din clasele de LM generice; totuși, fiind dat un LM acesta poate sa-și definească propria variantă de *sentanceProb*.

Tabelele hash, vectorii, sau alte structuri de date de bază au fost implementată de la zero, pentru mai multă viteză și pentru a fi mai compacte.

3.3 TEHNOLOGII PROGRAMARE

3.3.1 JAVA

Java este un limbaj de programare la nivel înalt dezvoltat de către Sun Mycrosystems. Java a fost numit inițial OAK, și a fost creat pentru a manipula dispozitivele portabile. Oak nu a avut succes așa că în 1995 Sun a modificat limbajul de programare pentru a putea lucra cu înfloritorul World Wide Web și, de asemenea, i-a schimbat numele în Java.

Java este un limbaj de programare orientat pe obiecte similar cu C++, dar simplificat pentru a elimina atributele de limbaj care ar cauza erori de programare des întâlnite. Fișierele cu codurile sursă Java (cele care au extensia .java) sunt compilate într-un format numit *cod-de-octeți* (fișiere care conțin extensia .class), și care vor fi executate de către un interpretor Java. Codul astfel compilat poate să fie executat pe majoritatea computerelor deoarece interpretoarele Java și mediile de execuție cunoscute ca *Mașini Virtuale Java (JVM)* există pentru majoritatea sistemelor de operare, inclusiv UNIX, Macintoch OS și Windows. *Codul-format-din-octeți* poate fi convertit în instrucțiuni de limbaj mașină de către un *compilator-exact (JIT).*

Java este un limbaj de programare utilizat în domenii foarte variate, cu un număr de atribute care fac un asemenea limbaj să fie potrivit folosirii mai ales pentru rețeaua Internet. Scurtele aplicațiile Java sunt numite apleturi Java și pot fi decărcate direct de pe serverul web, iar execuția lor se poate realiza direct pe calculatorul personal de către un browser web, cum ar fi Netscape Navigator sau Microsoft Internet Explorer.

3.3.2 JAVASCRIPT

JavaScript este un limbaj creat în scopul extinderii capabilităților navigatoarelor Web adăugându-le interactivitate. Cea mai importantă utilizare este posibilitatea de a creea funcții înglobate sau incluse în documentele XHTML, condiționate de arborele DOM al acestora având ca obiectiv realizarea unor funcții care nu pot fi îndeplinite folosind doar limbajul de marcare XHTML. O primă funcție importantă este deschiderea unei ferestre noi, ale cărei dimensiuni, aspect și poziție pot fi sub controlul programatorului. O altă funcție importantă este verificarea valorilor introduse într-un formular HTML, astfel încât acestea să fie validate înainte de a fi trimise înapoi serverului Web. Un ultim aspect, des utilizat, abordat de limbajul JavaScript este schimbarea sursei unei imagini, atunci când mouse-ul se află deasupra ei.

JavaScript a fost numit inițial Mocha și apoi LiveScript. Este un limbaj de *scripting* independent de platformă, implementat de către Brendan Eich de la Netscape și utilizat pentru prima oară în anul 1995 în browserul *Navigator 2.0B3.* În prezent JavaScript este o marcă înregistrată *Sun Microsystems.*

Așa cum s-a afirmat mai devreme JavaScript este un limbaj de scripting și se bazează pe conceptul de prototip. Ca sintaxă este asemănător cu limbajul de programare C. Dacă limbajul C folosește bibliotecile I/O, procesorul JavaScript (interpretorul JavaScript) se sprijină pe mediul

gazdă (*host environment*), cu alte cuvinte, cel mai frecvent este vorba despre browserul Web, care implementează obiecte de sine-stătătoare.

Limbajul JavaScript extins este alcătuit din: *Nucleul JavaScript,* care include elementele care stau la baza limbajului (operatori, expresii, instrucțiuni și obiecte predefinite), *JavaScript pe partea de client (CSJS)* care introduce, pe lângă nucleul principal, obiecte utilizate pentru controlul navigatorului, prototipului obiectului document (DOM) și, nu în ultimul rând, *JavaScript pe partea de server.* Acesta din urmă introduce elemente relevante pentru comunicația cu serverul (ex. Comunicarea cu baza de date sau manevrarea fișierelor). Versiunea 1.7 a JavaScript este inclusă în browserul *Firefox 2.0.*

JavaScript este utilizat pe un număr extins de platforme, inclusiv în afara Web-ului, cum ar fi companiile Adobe Acrobat și Adobe Reader, pentru fișierele PDF, în tehnologia Active Scripting (Microsoft), sau pentru Jscript a platformei .NET (Microsoft).

În mod uzual, codul JavaScript este înglobat direct în paginile XHTML, prin marcajul *script*, în felul următor:

<script type="text/javascript" src="Scripts.js">

</script>

O altă modalitatea de utilizare a codului JavaScript constă în salvarea lui într-un document cu extensia *js,* urmând ca acesta să fie inclus în pagină, ca în exemplul de mai sus.

3.3.3 HTML

Fundamental, când se privește o pagină web într-un navigator de Internet, se pot observa multe cuvinte. Însă, de cele mai multe ori paginile de internet conțin text stilizat și sunt îngrijit ornamentate. Actualmente, creatorii paginilor de Internet au acces la numeroase fonturi, culori și chiar diverse alfabete (ex. Spaniolă, Japoneză, Rusă), iar browserele pot, pentru cea mi mare parte din acestea să afișeze cu exactitate. Paginile web mai pot conține imagini, clipuri video și muzică de fundal. Se mai pot include diverse meniuri, casete de căutare sau legături pe care vizitatorii le pot folosi pentru a accesa alte pagini (fie pe același site sau pe altul). Unele dintre paginile disponibile pe Internet chiar permit clienților să-i modifice înfățișarea în funcție de preferințe sau necesități (ex. deficiențe de vedere, de auz sau daltonism). Deseori o pagină conține casete care posedă dinamicitate, în timp ce restul paginii rămâne static.

O pagină de Internet tipică depinde de o suită de tehnologii (cum ar fi CSS, JavaScript, Flash, AJAX, JSON) pentru a controla ceea ce văd utilizatorii, dar cel mai fundamental lucru, dezvoltatorii scriu aceste pagini în HTML, fără de care nu ar putea exita pagini de Internet. Pentru a afișa un astfel de conținut într-un dispozitiv pe partea de client, un browser citește codul HTML.

W3C (World Wide Web Consortium) și WHATWG (Web Hypertext Application Technology Working Group) menține standardele și specificările internaționale HTML. În 1980 Tim Berners-Lee, fizician la CERN (Organizația Europeană pentru Cercetare Nucleară), a inventat o modalitate prin care oamenii de știință să poată să facă schimb de documente prin intermediul Internetului. Înainte de acest eveniment, comunicarea pe Internetul era limitată la un simplu text, prin folosirea tehnologiilor precum e-mailul, FTP (Protocol al Transferului de Fișiere). HTML folosea un model păstrat pe un server central dar transferabil către o stație de lucru locală, vizibilă într-un browser, simplificând accesul la conținut sau făcând posibilă afișarea unui text mai amplu. HTML moștenește SGML, care este o sintaxă complexă pentru realizarea sau legarea conținutului (text sau grafice) în documente.

HTML este un limbaj de marcare. Termenul „marcare” a fost folosit de către editorii care își marcau manuscrisele (de obicei cu un stilou albastru) când dădeau instrucțiuni de verificări ulterioare. Marcarea însemnă, actualmente, ceva puțin diferit: un limbaj cu sintaxă specifică ce oferă instrucțiuni browserului web despre cum să afișeze o pagină. Încă o dată, HTML separă conținutul (cuvinte, imagini, etc.) din „prezentare” (instrucțiuni care afișează fiecare tip de conținut). HTML utilizează un set predefinit de elemente pentru definirea tipurilor de conținut. Elementele de dețin unul sau mai multe „tag-uri” exprimă un conținut. Tag-urilor le sunt atașate paranteze unghiulare, iar tag-ul de final începe cu semnul exclamării.

Majoritatea navigatoarelor permit utilizatoruluo să vizualizeze HTML-ul oricărei pagini de Internet. În firefox, spre exemplu, la apăsarea combinației Ctrl + U se permite vizualizarea codului sursă.

Limbajul HTML constă într-un set se elemente, ce definesc o semnificație semnatică a conținutului lor. Elementele includ două perechi de taguri și tot ce se află între acestea. De exemplu, elementul „<p>” indică un paragraf, <img> o imagine.

Anumite elemente au un înțeles precis, cum ar fi „aceasta este o imagine”, „acesta este un titlu”, sau „aceasta este o listă ordonată”. Altele sunt mai puțin specifice, cum ar fi „aceasta este o secțiune dintr-o pagină”, sau „aceasta este o parte a unui text”. Mai sunt, unele elemente sunt folosite doar din motive tehnice, ca de exemplu „acesta este o informație de identificare în pagină, așa că nu se va afișa”.

Elementele în HTML sunt aranjate ierarhic, <html> înconjoară restul documentului, iar elementele de <body> înconjoară conținutul paginii. Această strunctură poate fi gândită ca un arbore cu ramuri care „cresc” din trunchi (<html>). Structură ierarhică se numește DOM.

Tagurile sau etichetele sunt scrise într-un mod foarte simplu. Se poate redacta conținut HTML în orice editor (ex. Notepad, Notepad++, Sublime Text), dar majoritatea autorilor preferă să folosească un editor specializat care scoate în evidență sintaxa și arată DOM-ul.

Orice etichetă mai poate conține și o informație suplimentară ce poartă numele de atribut. Atributul constă într-un nume și o valoare. Câteva atribute dețin doar o valoare. Acestea reprezintă valori de adevăr și pot fi prescurtate doar prin specificarea numelui, sau lăsând valoarea atributului neinițializată. Prin urmare, cele trei exemple de mai jos au același înțeles.

<input required="required">

<input required="">

<input required>

Entitățiele în HTML sunt folosite pentru imprimarea caracterelor cu conținuut special în HTML. De exemplu, HTML interpretează simbolurile „mai mic decât” sau „mai mare decât” ca delimitări ale etichetelor. Dacă se dorește apariția caracterului propriu-zis într-un text, se pot folosi entități. Cele mai des utilizate entități sunt: &gt (>) &lt (<) &amp (&) &quot (”).

3.4 PROTOCOALE DE COMUNICAȚIE

3.4.1 REST

Multe servicii de internet moderne ignoră standardele web și își dezvoltă propriile interfețele pentru a-și face publice aplicațiile. Aceste abordări reduc interoperabilitatea și măresc latența rețelei, reducând și scalabilitatea serviciului. Rețeaua a crescut de la cateva zeci de cereri într-o zi la un milion de cereri pe oră fără o reducere semnificativă a performației. Cu aceeași arhitectură care stă la baza serviciilor *Web către Web* se pot improviza aplicațiile existente

precum și cele din viitor. REST este un model idealizat al interacțiunilor dintre aplicațiile web și a devenit temelia arhitecturii web, fiind construit pentru a întâmpina nevoile sistemelor distribuite hiper-media, punându-se accentul pe scalabilitate, pe generalizarea interfețelor, lansarea independentă permițând, de asemenea, componentelor intermediare să reducă latența rețelei.

Pentru a construi o aplicație web cu grad ridicat de interoperabilitate nu trebuie trecute cu vederea standardele. Rețeaua – cea mai mare aplicație web disponibilă – nu este definită printr-o implementare particulară, este definită de anumite interfețe standard și protocoale. Interoperabilitatea duce la utilizare crescută, astfel că scalabilitatea este încă o mare problemă în cadrul sistemelor distribuite.

Există multe standarde la ora actuală pentru aceleași domenii, fiecare dintre acestea nu propriile limite și avantaje. În industrie s-au dezvoltat mai multe modalități competitive care să permită dezvoltatorilor să construiască servicii web cât mai simplu cu putință. Majoritatea se bazează pe RPC (Apeluri de Procedură la Distanță) pentru a trimite mai departe mesajele între serviciile implicate. Prin folosirea modelelor RPC, serviciile web pot să ducă la defecțiuni severe din punct de vedere al performației și al scalabilității.

Stilul arhitectural care stă la baza rețelei Internet se numește REST (Stări de Transfer Reprezentaționale). REST răspunde nevoii de Internet Engineering Task Force (IETF) pentru un model al funcționării corecte a rețelei Internet. Este un model idealizat al interacțiunilor dintr-o aplicație web globală.

Roy T. Fielding definește REST-ul ca un set coordonat de constrângeri arhitecturale ce încercă minimizarea latenței și a comunicației în interiorul rețelei, maximizând în același timp independența și scalabilitatea implementării componentelor. REST permite stabilirea și refolosirea interacțiunilor, înlocuirea dinamică a componentelor și realizarea acțiunilor de către intermediari, prin urmare, ridicându-se la nevoile sistemului hiper-media numit Internet.

REST distinge trei clase de elemente arhitecturale: datele, conectorii (elementele de conexiune) și elementele de procesare (componentele). Aspectul cheie al REST-ului este starea datelor, componentele acestuia comunică prin transferul reprezentărilor stărilor curente sau dorite a elementelor de informație.

Resursele realizează o teoretizare a principiului REST. Orice deține un nume se cheamă resursă. O resursă este o hartă computațională a unui set de entități, doar semnatica resursei este statică, entitățile, însă, se pot schimba în timp. Aceasta înseamnă că entitatea din spatele resursei se poate schimba în timp. Acestă noțiune permite autorului să referențieze un concept în locul unei singure reprezentări.

REST folosește identificatori de resursă pentru a distinde între resurse. În mediul de dezvoltare al Internetului, identificatorul ar fi URI (Identificatorul Uniform de Resursă) definit în FRC 2396.

Toate componente REST efectuează acțiuni asupra resurselor reprezentate. Reprezentările captează starea intenționtă a resursei și pot fi tranferate între componente diferite. Ele constau într-o secvență de biți (conținutul), metadata reprezentativă (care exprimă conținutul) și metadata ce descrie metadata (sume hash sau Controlul Memoriei Cache).

REST nu este legat de un format al datelor specific atâta timp cât toate componentele pot procesa date. O memorie cache intermediară, de exemplu, nu are nevoie să cunoască semnificația datelor, doar în cazul în care conținutul sau răspunsul este de memorat sau nu.

Formatul datelor unei reprezentări este numit *tip media.* Câteva tipuri media pot fi procesate de către computere, altele sunt realizate cu scopul de a fi lizibile pentru cititorul uman și puține pot fi procesate automat și vizualizate de către utilizatori. Proiectarea tipului media are un efect

imediat asupra performaței sistemului. Orice dată trebuie să fie recepționată înainte de procesarea cererii. Un tip media cu date importante la începutul fluxului de date poate fi procesat în timp ce este recepționat, și, prin urmare, va scădea timpul global de răspuns la cerere. De exeplu, un browser web capabil să desfășoare pagina web în timp ce o primește furnizează o performanță vizibil mai mare față de un browser fără acea capabilitate.

În serviciile web moderne o reprezentare răspândită este XML (Limbaj de Marcare Extensibil) definit de către World Wide Web Consortiun (W3C), alte reprezentări potrivite numai pentru utilizatorul uman ca și HTML, JPEG sau PDF sunt, de asemenea, foarte des utilizate. Metadatele de tipul Control-al-Memoriei-Cache, timpul de ultimă modificare și tipul media pot fi folosite pentru a alege în mod dinamic între diferite reprezentări posibile și pentru îmbunătățirea scalabilității prin difinirea explicită a reprezentărilor. Conectorii administrează comunicația în rețea pentru o anumită componentă. Aceștia prezintă o interfață abstractă pentru comunicația între componente ducând la separarea aspectelor problematice, la ascunderea implementării de la baza sistemului, sporesc simplitatea și asigură substituibilitatea. Fiecare interacțiune REST nu este o stare în sine, fiecare cerere conține informația necesară pentru ca un conector să înțeleagă cererea, independent de alte cereri care au precedat-o.

Toate aceste constrângeri duc la următoarele avantaje:

- Conectorul nu este nevoit să rețină starea aplicației, ceea ce însemnă că fiecare componentă va fi simplificată și scalabilă.
- Cererile vor fi servite în paralel deoarece nicio interacțiune semnatică nu mai trebuie înțeleasă.
- Cererile pot fi înțelese izolat pentru a simplicarea orchestrării și rearanjării dinamice a serviciului
- Se permite memorarea temporară.

REST încapsulează diverse activități pentru accesarea și transferul reprezentărilor în diferite tipuri de conectori:

- Clientul – trimite cereri și primește răspunsuri;
- Serverul – ascultă cererile și trimite răspunsurile;
- Memoria cache – poate fi localizată în conectorul clientului sau serverului pentru a salva răspunsuri care necesită spațiu de memorare, aceasta poate fi, se asemenea, împărțită între mai mulți clienți;
- Rezolvator – transformă identificatorul resursei în adrese de rețea;
- Tunelul – susține cererile, orice componentă își poate schimba starea din activă în comportament de tunel;

Orice componentă poate implementa mai mult de un tip de conector. Componentele REST sunt identificate prin rolul în cadrul aplicației. Utilizatorul utilizează un conector tip client ce este sursa definitivă pentru reprezentarea resurselor sale. Fiecare server furnizează o interfață generică serviciilor sale ca și o ierarhie a resurselor. Componentele intermediare au un comportament combinat între client și server pentru a transmite mai departe – cu translații posibile – cererile și răspunsurile. Exemple de compnente sunt Apache httpd ca server de origine, orice browser web pe post de utilizator și Squid ca Proxy sau Gateway. REST își definește arhitectura punând restricții asupra interacțiunii dintre componente, în loc să se definească o altă topologie de componente.

Pentru a adânci înțelegerea arhitecturii REST se va dezvolta o aplicație web simplă cu scop educativ. Serviciul va oferi urmatoarea funcționalitate: utilizatorul poate încărca o fotografie, metadata poate fi inserată în fotografie, fotografiile și metadatele pot fi șterse, se poate crea o listă a fotografiilor, iar poza și metadata pot fi recuperate.

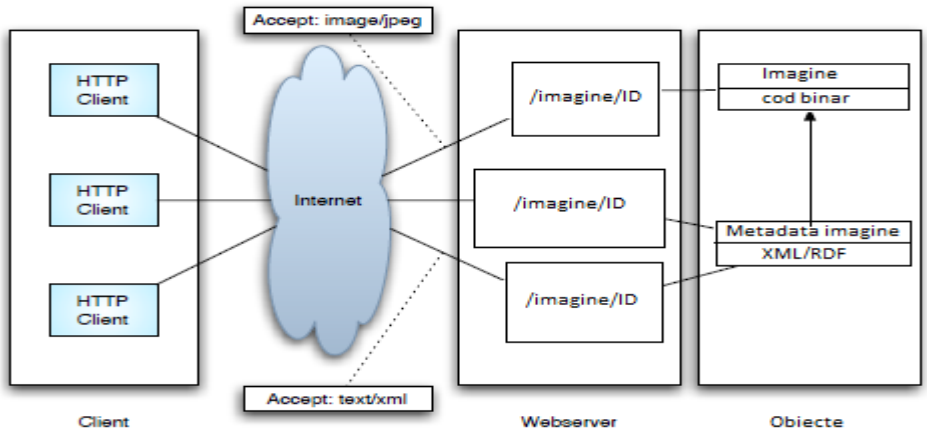

Fig. 2 O privire de ansamblu a aplicației web

A se observa că interfața nu ar trebui să expună felul cum obiectele sunt salvate! Stratul „Obiect" trebuie să conțină fișiere simple precum și o bază de date.

Abstractizarea în REST este resursa, deci un bun început al aplicației este identificarea resurselor și anume: imaginea și colecția de imagini. Fiecare resursă are asociată o reprezentare: imaginea se reprezintă în binar sau prin XML, iar colecția de imagini prin XML. Resursele, în acest caz sunt adresabile prin URI. Doar resursele pot fi adresate, nu și reprezentările lor, deci metadata trebuie despărțită de imaginea în sine.

- Imaginea: /imagine/ID
- Colecția: /imagine/index

În loc de a adresa explicit metadata și reprezentarea binară a resursei, clientul ar putea să utilizeze negocierea conținutului pentru a determina ce reprezentare ar trebui întoarsă. *Accept: text/xml* din antetul HTTP al cererii va primi o reprezentare XML a resursei, iar *Accept: imagine/jpeg* solicită o reprezentare binară. HTTP definește un set de metode, în acest exemplu se vor primi cereri de tip GET, PUT, POST și DELETE. Metodele suplimentare precum ANTET sau OPȚIUNI pot fi folosite pentru a angaja o memorie cache îmbunătățită (ANTETUL), și să returneze o descriere a serviciului sau un URI-ului cerut.

PUT este folosit pentru a încărca o noua imagine pe server. Această acțiune solocită o autentificare prin HTTP AUTH. Cererea PUT restituie codul răspuns *201 Creare* precum și un URI către resursa creată. PUT poate fi executat din */imagine*. Definiția HTTP-ului susține că toate metodele, în afară de POST, sunt la fel de puternice, așa că ID-ul generat pentru imagine trebuie să fie stabil!

Cerere:

PUT /imagine HTTP/1.1

AUTORIZARE: dGVzdDp0ZXN0MQ==

GAZDĂ: localhost:2000

MĂRIMEA CONȚINUTULUI: 13077

Răspuns:

HTTP/1.1 201 Creare

CONEXIUNEA: realizată

DATA: Wed, 23 Feb 2005 12:18:23 GMT

SERVER: WEBrick/1.3.1 (Ruby/1.8.2/2004-12-25)

MĂRIMEA CONȚINUTULUI: 44

http://localhost:2000/imagine/11091611039546

POST este folosit pentru a adăuga mai multe meta-informații resursei adresate. De exemplu datele GPS precum longitudine și latitudine pot fi adăugate cu ușurință unei resurse. POST se poate executa în /imagine/ID. Acesta întoarce reprezentarea actualizată a meta-informației imaginii, dar necesită autentificare.

DELETE poate fi folosit pentru a șterge o resursă. Poate fi aplicat pe /picture/ID și solicită autentificarea. Dacă serverul acceptă cererea, răspunsul este codul *202 Acceptare.*

GET este folosit pentru recupera o reprezentare a unei resurse specifice. Nu solicită nicio autentificare și se poate aplica pe /imagine/index pentru a obține o listă cu toate imaginile disponibile, metoda GET aplicată asupra /imagine/ID cu *Acceptare: text/xml* pentru a obține metadata imaginii și *Acceptare: imagine/jpeg* pentru a obține o reprezentare binară.

OPTIONS poate fi folosit pentru întoarcerea către utilizator a unei descrieri a serviciului accesat.

3.4.2 JSP

Paginile JavaServer și Servlet-urile sunt tehnologii complementare care produc pagini web în mod dinamic prin intermediul limbajului de programare java. În timp ce servlet-urile reprezintă piatra de temelie pentru programele java pe partea de server, nu sunt mereu cea mai eficientă soluție luând în considerare timpul de dezvoltare. Codarea, lansarea sau rezolvarea erorilor unui Servlet pot fi considerate o sarcină plictisitoare. Prin modificarea unei greșeli gramaticale sau a unei etichete necesită o procesare îndelungată asupra metodelor de print() sau println(), recompilarea Serlet-ului sau reîncărcarea aplicației web. Primul schelet pentru paginile JavaServer a fost lansat în 1999. Opțional, JSP a fost modelat după alte tipare pe partea de server pentru a realiza o metodă simplă de îmbinare a metodelor de marcare care au un caracter static cu tehnologiile de programare dinamice. Când o cerere este făcută în ptivința unui conținut al unei pagini JSP, un container interpretează JSP, execută orice fel de cod încorporat și trimite rezultatul în răspuns.

JSP a fost îmbunătățit de mai multe ori de la prima lansare, de fiecare dată primind o nouă funcționalitate, și actualmente este la versiunea 2.0. Specificațiile JSP se pot consulta pe paginile de Internet ce aparţin companiei Sun Microsystems.

O pagină JSP de bază constă într-un text uniform, conține etichete și profită de avantajul că rulează programe dinamice sau scripturi. JavaBean nu este definit pentru JSP, dar JSP oferă suport pentru utilizarea acestora. Există diverse mecanisme de legătură dintre o marcare statică și un cod java. Aceste mecanism este de departe punctul forte al paginilor JSP și poate fi folosit în locul claselor încorporate alcătuite din cod java.

JSP include mecanisme pentru definirea atributelor dinamice pentru tag-uri obișnuite. Orice script poate fi folosit în acest scop, de obicei este implementat un cod java. Pentru a înțelege mecanismele pe care se bazează paginile jsp trebuie mai întâi înțeles ciclul de viață a acestor construcții. JSP se ghidează după trei etape ale existenței: inițializarea, serviciul, și distrugerea. JSP a fost special conceput pentru a simplifica sarcina de creare și producere de text cu ajutorul

obiectelor *HttpServlet*. Nu există nicio diferență de principiu între paginile JSP normale și paginile HTML. Toate JSP-urile sunt gândite special pentru a putea fi folosite împreună cu paginile HTML și să genereze conținut dinamic pentru rețeau globală Internet. Simpla metodă _jspService este, de asemenea, responsabilă pentru generarea răspunsurilor pentru toate cele șapte metode HTTP. Din motive practice, un dezvoltator JSP nu este nevoit să dețină cunoștințe depre HTTP, ci doar să aibă o bază pentru realizarea codurilor în limbaj java.

O diferență inportantă între JSP și servlet-uri este că metodele de codare sunt diferite. Un exemplu simplu de pagină JSP (salutare.jsp) este următorul:

```
<html>
<head>
<title>Salutare!</title>
</head>
<body>
<h1>Salutare!</h1>
</body>
</html>
```

Redactarea unei pagini JSP este la fel de simplă ca realizarea paginilor HTML. În comparație cu folosirea metodelor print() și println() din Servlet-uri, abordarea JSP este mult mai simplă. De aceea JSP este considerat ca o metodă rapidă de a crea Servlet-uri care produc text.

Lansarea unui cod JSP este de asemenea foarte facilă, o aplicație web lansează automat orice pagină JSP către o extensie a URL-ului care se potrivește cu numele paginii JSP. Ceea ce nu este atât de evident dar este foarte important de înțeles este că pagina salutare.jsp este compilată într-un cod echivalent pentru Servlet. Acest lucru se realizează în etapa de translație și este realizat automat de către container. Translația JSP este foarte importantă deoarece se explică exact cum o pagină JSP este transformată în cod java. JSP-ul transformat în cod java poate fi găsit în partea din dreapta a containerului particular. Serverul Tomcat păstrează acest cod în directorul /work. Codul generat nu este niciodată atractiv, și nici nu are aceeași denumire. Programatorul nu trebuie să se chinuie cu înțelegerea acestui cod generat. Cel mai important aspect care trebue înțeles este că pagina JSP este lăsată în grija containerului, iar transformarea acesteia în servlet se realizează în spatele scenei.

În cadrul paginilor JSP totul este împărțit în două mari categorii: elemente și date ce aparțin unui anumit tipar. Elementele reprezintă partea dinamică a paginilor JSP. Datele sunt binare. Acestea pot fi ușor împărțite pe categorii, de vreme ce tot textul plasat arbitrar pe o pagină JSP este destinat livrării directe către client. Conceptele și tiparul unei pagini JSP sunt foarte importante deoarece acestea dictează unde, când și ce text va fi poziționat în pagina web. Exemplul salutare.jsp a fost în întregime un tipar de text. Servlet-ul corespunzător generat din salutare.jsp este tratat ca un conținut static și este trimis ca răspuns. Elementele unei pagini JSP, pe de altă parte, nu sunt trimise direct clientului. Un element este interpretat de containerul JSP și definește acțiuni speciale care ar trebui realizate atunci când se generează un răspuns. Ele se împart în trei categorii: elemente de scripting, directivele și acțiunile.

O pagină JSP se poate redacta în două moduri: modul normal, ușor de interpretat de către programator și modul compatibil XML care facilitează comunicația cu diferite utilitare. Cât despre elementele de scripting, acestea sunt și ele de trei tipuri: scriplet, expresie și declarație. Scriptlet-urile furnizează o metodă directă de inserție a biților de cod java între bucăți de text HTML. Un scriplet este definit la început de <% și se încheie cu %>, având inserat cod între acele simboluri. Folosind java, scriptul este identic cu un cod java normal dar fără necesitatea de a declara clasele sau metode. Scripleturile sunt foarte utile în a realiza funționalități de nivel inferior cum ar fi iterațiile, buclele, sau instrucținile condiționale. Din mai multe motive, însă, scriptlet-urile ar trebui evitate. Aceasta se datorează faptului că, pe măsură ce numărul scriplet-urilor crește, va fi cu atât mai greu să se înțeleagă și să se mențină funcțională o pagină JSP.

<html>

```
<head>
<title>Loop Example</title>
</head>
<body>
<% for (int i=0; i<5;i++) { %>
Repeated 5 Times.<br>
<% } %>
</body>
</html>
```

Expresiile diferă ca principiu și oferă o metodă ușoară de a trimite date în mod dinamic către client. O expresie începe cu <% și se încheie cu %>. O expresie diferă de un scriplet printr-un semn „=” care trebuie să apară imediat după start. Expresiile mereu trimit un element de String către cliect dar ele nu returnează obligatoriu tot un element de tip String. Obiectul returnat deține propria metodă toString pentru a determina valoarea expresiei. O utilizare a expresiilor ar putea fi metoda de iterație printr-o colecție de valori. Scriptlet-ul oferă bucla, iar expresiile și conținutul static sunt folosite pentru a trimite informație.

```
<% String[] strings = new String[4];
strings[0] = "Alpha";
strings[1] = "in";
strings[2] = "between";
strings[3] = "Omega";
for (int i=0; i<strings.length;i++) { %>
String[<%= i %>] = <%= strings[i] %><br> <% } %>
```

Declarațiile sunt ce de-a treia și ultimul element de scripting disponibil pentru o pagină JSP. O declarație seamănă cu un scriptlet, dar codul încorporat în HTML este plasat în exteriorul metodei _jspService(). Din acest motiv, codul care alcătuiește o declarație poate fi folosit pentru a crea metode noi și clase java, dar trebuie multă atenție în realizarea codului deoarece nu este asigurată siguranța funcționării firelor de execuție, doar dacă propgramatorul însuși va avea grijă de acest aspect.

Directivele reprezintă mesaje trimise către un container JSP. Nu se trimite nimic către client, dar sunt foarte utile în definirea atributelor paginii, alegerea bibliotecilor adecvate și stabilirea includerii unor alte clase. Toate directivele folosesc următoarea sintaxă:

```
<%@ directive {attribute="value"}* %>
```

3.5 SERVERE DE APLICAȚII JAVA: APACHE TOMCAT

Scopul suprem al serverului Tomcat este să furnizeze suport conform standardelor pentru servleturi și JSP. Menirea servleturilor și paginilor JSP este să genereze conținut web cum ar fi pagini HTML sau fișiere GIF la cerere, prin utilizarea datelor variabile. Conținutul web care este generat la cerere este menit să fie dinamic, spre deosebire de conținutul web care nu se schimbă niciodată și se numește static, cum ar fi imaginile sau CSS-urile.

În timp ce Tomcat este capabil să servească atât conținut static cât și dinamic, nu este atât de rapid și nici nu are atât de multe caracteristici ca serverele web create special să poată servi conținut static. Ar fi posibil pentru Tomcat să fie extins pentru a suporta mai multe funcționlități, dar asta ar dura prea mult timp. Cel mai popular server de la Apache a rămas subdezvoltat mai mulți ani la rând. Mai mult, deoarece majoritatea serverelor sunt construite folosind limbaje de

nivel jos cum ar fi C și profită de trăsăturile specifice platformei, este puțin probabil ca Tomcat (o aplicație java 100%) ar putea acționa la fel de bine ca niște asemenea produse.

Este recunoscut faptul că Tomcat se bucură de o relație sinergetică cu serverele web convenționale, cele mai vechi versiuni ale lui Tomcat includ un conector care permite ca Tomcat și Apache să lucreze impreună. În astfel de relații, Apache primește toate cererile HTTP făcute către aplicația web. Apache recunoaște apoi care cereri sunt destinate Servlet-urilor/JSP-urilor, și duce aceste cereri către Tomcat. Tomcat împlinește cererea și trimite răspunsul înapoi către Apache, care înapoiază răspunsul către destinatar.

Conectorul Apache a fost inițial de o importanță crucial pentru Tomcat 3.x, deoarece acesta suportă atât conținutul static, dar implementarea protocolului HTTP era cumva limitată.

Începând cu seriile 4.x, Tomcat oferă funcționalități pentru o implementare mai complexă a HTTP-ului și un support mai bun pentru a putea servi un conținut static, și ar trebui să fie suficient el însuși pentru personae care nu caută o performață crescută, dar au nevoie de protocolul HTTP în realizarea aplicațiilor lor. Totuși, așa cum s-a mai menționat, Apache și alte servere web vor avea, cel mai probabil, o performanță superioară și mai multe opțiuni când vine vorba de servirea conținutului static și comunicarea dintre aplicații și clienți prin HTTP. Pentru acest motiv, orice folosește Tomcat pentru trafic mare de date ar trebui să ia în considerare colaborarea dintre Tomcat și un alt server web.

Implicit, Tomcat rulează pe portul 8080. Orice cerere care este trimisă către un server HTTP de pe portul 80, și orice cerere către portul 8080 sunt trimise către Tomcat. Se poate crea codul HTML al unei aplicații web pentru a-și solicita resursele statice de pe serverul web de pe portul 80.

Arhitectura unei aplicații web

Setul tuturor servleturilor, paginilor JSP, și a altor fișiere care sunt în relații logice constitue o aplicație web. Specificațiile servlet-ului definesc o ierarhie de directoare, în care toate fișierele trebuie să fie prezente. Aceste fițiere sunt descries în tabelul de mai jos:

Calea relativă	Descriere
/	Rădăcia aplicației web: toate fișierele accesibile public sunt plasate în acel director. Exemple includ HTML, JSP, sau fișiere GIF.
/WEB-INF	Toate fișierele din acest director și toate subdirectoarele nu sunt accesibile public. Un singur fișier, *web.xml,* care se numește *descriptor de lansare în execuție,* conține opțiuni de configurare pentru aplicația web. Diversele opțiuni pentru acest descriptor sunt definite de către Servlet API.
/WEB-INF/classes	Toate clasele aplicației web se găsesc aici.
/WEB-INF/lib	Clasele acestea pot fi arhivate ca fișiere JAR și plasate în acest director.

Tab. 12 Ierarhia directoarelor unei aplicații web

Toate containerele de tip Servlet solicită folosirea acestei ierarhii de directoare. Mai mult, datorită faptlui că locația și a specificațiile descriptorului (*web.xml*) sunt setate explicit, aplicația web are nevoie doar de o singură configurare. Descriptorul de lansare definește opțiuni cum ar fi ordinea în care se vor încărca servlet-urile de către container, parametrii care pot fi trecuți mai departe către servlet la pornire, care URL este folosit pentru anumite servleturi, ce restricții de securitatea are aplicația și așa mai departe.

Pentru a facilita distribuția, toate fișierele din ierarhia descrisă în **tabelul precedent** pot fi arhivate într-un WAR (Web ARchive). Administratorii de server pot să pună fișierul WAR într-un director specificat de către containerul Servlet-ului, iar containerul va avea grijă de restul aspectelor de aici înainte.

CAPITOLUL 4

EXPERIMENTE. FUNCȚIONALITATE. SCENARII DE UTILIZARE

4.1 SERVICIU WEB ACCESIBIL DE ORICE APLICAȚIE

Trăim în epoca informațională, iar orice creație nou apărută în spațiul virtual trebuie să se supună unor convenții care să înlesnească progresul. Aceste principii feresc programatorii de a „reinventa roata” de fiecare dată când își construiesc propriile sisteme. Se folosesc de alte utilitare, API-uri, sau programe realizate, cu multă trudă, de către alții, care au fost de acord să-și facă publice codurile. Prin urmare, este necesară o comunicație standardizată pentru a stabili conexiuni cu alte servicii deja existente pe Internet.

Sistemele și infrastructurile sunt în continuă dezvoltare pentru a putea susține serviciile web. Ideea principală este de a încapsula funcționalitatea unui proces printr-o interfață adecvată și de a fi scoasă în evidență prin intermediul unui proces web. Viziunea serviciilor web este de permite partenerilor anonimi să-ți facă publice capabilitățile și să se angajeze în comunicații, să interacționeze cu orice alt partener și să permită refolosirea codurilor sursă. În timp ce tehnologiile web au avut o influență puternică asupra potențialului infrastructurii web prin furnizarea accesului programatic la informație și servicii, acestea întâmpină o piedică

semnificativă prin lipsa abstactizărilor legate de înțelegerea între mașinile pe care au fost create sistemele şi pe cele pe care vor fi nevoite să lucreze. Tehnologiile web ar trebui mai bine integrate, organizate și cu timp de căutare mai scurt. Pentru o detaliere mai bună a funcționalității serviciilor web este necesar un exemplu. Dintre cele mai concludente este un serviciu web de restaurare a diacriticelor. Aceasta are la bază un sistem de inserție bazat pe principiul dezamguizării cuvintelor. Serviciul constă în faptul că se respectă principiul interoperabilității cu alte aplicații importante la momentul actual, precum extragerea de informaţii, traducerea automată, colecţionarea de texte, construirea dicţionarelor electronice şi multe altele. Corectarea erorilor ortografice poate să aibă un impact major asupra calității rezultatelor obţinute în aceste aplicaţii. Așadar serviciului îi crește valoarea mai ales când va fi integrat în alte sisteme.

O aplicație care colecționează date poate, spre exemplu, să comunice prin intermediul protocolului HTTP pentru a restaura diacriticele în textele descărcate de pe Internet. Din datele oferite de Horia Cucu, Andi Buzo, Laurent Besacier și Corneliu Burileanu în „SMT-based ASR domain adaptation methods for under-resourced languages: Application to Romanian”, se poate constata că pentru un sistem de recunoaștere automată a vorbirii se poate scădea rata erorii la nivel de cuvânt (WER%) de la 60% la aproximativ 30%.

Sistemul dezvoltat de comitetul de cercetare de la laboratorul SpeeD a dezvoltat un sistem de restaurare a diacriticelor, ce folosește un model de limbă bazat pe 5-grame. Acesta dă rezultatele cele mai bune la ora actuală pentru textele din limba română. 98% din diacritice sunt restaurate corect, mai multe detalii despre experimentele care au fost efectuate au fost prezentate în capitolul I. Diversitatea aplicațiilor care ar putea beneficia de pe urma acestui sistem a determinat dezvoltarea unui serviciu web care să poată fi accesat de oriunde prin protocolul HTTP, eliminându-se și dependența de platforma hardware. Tot ce este necesar pentru ca să poată funcționa sistemul este existența unei mașini virtuale java (JVM) pe care să poată rula codul aplicației.

Acestea fiind spuse, proiectul de față a pornit pas cu pas de la o sarcină simplă de inserție a diacriticelor unui text mai amplu. Programul avea erori datorită sitemului de preprocesare a textului care se bloca din cauza unor caractere speciale. Textul necesita o preprocesare datorită faptului că sistemul de restaurare lucrează cu texte uniforme fără semne de punctuație, majuscule sau multe spații goale (linii goale sau tab-uri suplimentare). Așadar acestea au trebuit să fie eliminate, dar nu oricum, ci prin realizarea unei evidențe a poziției acestor caractere nedorite în text. Această preprocesare a fost realizată cu o bibliotecă în java creată de Horia Cucu în 2011 și returnează textul monoton și încă două fișiere ce conțin semnele de punctuație împreună cu poziția și respectiv literele mari. După preprocesarea, procesul de inserție se realizează printr-o singură comandă în Linux cu anumiți parametri despre care vom vorbi mai târziu. Astfel, rezultatul acestui proces, care are o mică întâziere, trebuie supus și el unei postprocesări care să înlocuiască, la locul potrivit, caracterele cu probleme. Această mică sarcină fiind împlinită, s-a dorit o aplicație care să realizeze automat aceste operații, iar fișierul de intrare să fie dat ca parametru.

Pentru a nu fi nevoie de o a treia aplicație care să exemplifice funcționare serviciului web a fost creată și o aplicație web cu scop demonstrativ. Aplicația s-a dorit să poată fi accesată de oriunde din Internet, pe site-ul http://www.dev.speed.pub.ro. Funcționalitatea aplicației se putea rezuma în momentul incipient la o pagină HTML și la un protocol de comunicație, care să permită extragerea textului introdus într-o casetă text, să-l proceseze și să întoarcă rezultatul către utilizator.

4.2 APLICAȚIA WEB – DEMONSTRAȚIE A SERVICIULUI

Pentru crearea aplicației au fost gândite două metode, dar din motive mai jos menționate, doar varianta a doua a sistemului se va ridica la nivelul așteptărilor, adică să îndeplinească rolul de serviciu web accesibil tuturor utilizatorilor.

La început, o implementare simplă a fost prin intermediul paginilor JSP. Aceste pagini, așa cum au fost detaliate în capitolul anterior, sunt exprem de bine primite de către aplicațiile care nu au funcționalități care permit navigarea către multe alte ferestre înrudite, sau care răspund unor cereri foarte diverse. Acestea din urmă necesită încărcarea unei pagini JSP de ficare dată când se solicită un serviciu. Operațiunea este consumatoare de timp, sau mai rău, dacă nu se solicită altă fereastră dar solicităm anumite operații într-un formular din pagina respectivă, pagina JSP se încarcă în întregime cu datele actualizate de fiecare dată. Acest mare dezavantaj face ca arhitectura bazată pe pagini JSP să fie foarte complicată și să necesite multă muncă de redactare. Avantajul acestei tehnologii se bazează pe faptul că orice funcție care se dorește responsabilă de o anumită parte dintr-o pagină web este ideal poziționată cât mai aproape de locul cu pricina. Astfel, este foarte simplu ca într-un să introducem cod java fără a face o clasă separată ci doar prin inserarea tab-urilor de <% *cod java* %>.

Odată aleasă modalitatea de obținere a datelor și a răspunsului prin pagini JSP s-a ajuns la concluzia că au fost necesare doar două astfel de pagini care să comunice între ele prin apăsarea butonului de „submit”. Preluarea textului și transmiterea acestuia pentru procesare se face în cadrul paginii JSP (care este o pagină HTML, numai că deține cod java), iar acest text, transformat în String, este transmis ca parametru unei clase java cu următoarele funcționalități:

- Realizarea unui fișier, solicitat de operația de preprocesare;
- Preprocesarea
- Inserarea în clasa java a unui script în linie de comandă Linux pentru a porni sistemul de inserție a diacriticelor și care va întoarce fișierul restaurat.
- Postprocesarea textului restaurat.
- Returnarea unui fișier final.

În altă parte a paginii web, într-o zonă de text diferită, va fi apelată, în pagina JSP, o citire a fișierului de ieșire și o scriere automată a acestui text.

Acest proiect conține, pe lângă ce două pagini JSP o clasă java, o clasă CSS pentru înfrumusețarea aplicației printr-un albastru plăcut aplicat titlului și, nu în ultimul rând, două funcții javaScript care să aplice funționalitate butoanelor de încărcarea a unui fișier de pe disk și a salvării fișierului rezultat. Butonul de încărcare deschide o fereastră care să permită utilizatorului să caute pe calculatorul propriu după fișierul care necesită restaurarea, iar butonul de salvare realizează o simplă descărcare.

Sistemul astfel construit nu se supunea regulilor comunității open-source, astfel că pentru a putea face disponibil sistemul și altor aplicații și ca acesta să poată fi accesat de oriunde din Internet era necesară o altă abordare. Așa că următorul pas a fost realizarea variantei a II-a a sistemului prin implementarea cu REST API.

Serviciul REST necesită o abordare mai complicată, dar, odată scheletul arhitecturii fiind realizat adăugarea de noi funcționalități este mult mai facilă.

O primă abordare este stabilirea URL-ului ca metodă de reprezentare a procesului de restaurare. În acest moment nu se mai folosesc pagini JSP ci doar o singură pagină HTML, care reprezintă pagina de start a serviciului. Utilizatorul va introduce textul sau îl va încărca de pe disk, căci funcționalitățile butoanelor de încărcare și descărcare (din codul javaScript) au rămas identice. Ce s-a mai modificat în pagina HTML sunt atributele containerului în care sunt așezate cele două câmpuri de text. Pentru mai multă precizie voi prezenta o comparație a celor două:

<form action="da.jsp" id="container"> și

<form name = "myform" method = "POST" enctype = "multipart/form-data" id = "info" onsubmit = "return false">

Cele două <form> au atribute diferite. Pentru primul caz se specifică numele altei pagini JSP care să realizeze operația de restaurare, iar în cazul al doilea este specificată metoda principală care va fi realizată în cadrul acelei bucăți din pagina HTML, adică metoda POST, pentru că se va trimite către server textul care se dorește a fi procesat.

Folosind principiile REST, se poate observa că metoda invocată în interiorul <form> este POST. Odată apasat butonul de „Submit" se va creea un URL format dintr-o rădăcină, iar ca terminație vor fi transmise variabila *textIntrare* împreună cu valoarea sa, chiar textul dorit a fi procesat. Odată format acest URL el este apelat și se face o trimitere către metoda java care este responsabilă cu procesarea textului:

```
termination = "?inputText=" + inputText;
var baselink = "http://dev.speed.pub.ro:10082/DiacriticsRESTorer

var xmlHttp = new XMLHttpRequest();
xmlHttp.open("GET", baselink + termination, false);
xmlHttp.send();
var baselink = "http://dev.speed.pub.ro:10082/DiacriticsRESTorer
```

Clasa java importă următoarele pachete:

```
import javax.servlet.http.HttpServletRequest;
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
```

Clasa javax.ws.rs conține interfețe de nivel înalt și adnotări pentru a creea resurse și servicii REST. Metoda realizează o cerere de tip GET de la calea specificată de funcția specială @Path(). Cererea GET primește un parametru de intrare dat de procedura @Consumes() o resursă web și returnează un alt obiect în formatul stabilit de metoda @Produces(). Metodele care se pot creea cu ajutorul acestor clase java sunt metode care au redactate deasupra lor parametrul @GET, @POST, sau @PUT și care primesc ca parametru un obiect de tipul `@Context HttpServletRequest`. Instanța „request" deține metoda `request.getParameter()` care încarcă într-o variabilă de tip String conținutul apelului metodei GET.

Odată cu stabilirea conexiunii dintre pagina HTML și codul java de procesare automată a textului, lucrurile decurg de la sine cu o oarecare lejeritate. Singurul lucru care se modifică în clasa contruită pentru varianta I a sistemului de restaurare este faptul că această clasă de procesare va întoarce un rezultat de tip String din fișierul de ieșire:

```
    outputString = readFile(_pathcorpusOut);
    return outputString;
```

Acest rezultat este privit ca un răspuns la cererea făcută către sistem şi prin intermediul funcţiei javaScript, în locul potrivit, se va afişa:

```
var responseText = xmlHttp.responseText;
document.getElementById("outputText").value=responseText;
```

O altă problemă care intervine în crearea unui serviciu web este faptul că acesta are nevoie ca alte resurse externe codului să fie disponibile tuturor clienţilor care folosesc sistemul. Printre aceste resurse se pot intui uşor fişierele text. Acestea sunt citite din mediul de programare din care se rulează aplicaţia. Deoarece aplicaţia va fi accesibilă oriunde pe Internet, aceste fişiere vor fi localizate pe serverul care susţine execuţia ei. Serverul care a fost ales este Apache Tomcat. Fişiere sunt următoarele: de intrare, de preprocesare, cel cu semnele de punctuaţie, cel cu majuscule, apoi cel returnat de către programul disambig şi, nu în ultimul rând, documentul de ieşire. Un aspect important care necesită o abordare specială reprezintă folosirea celor două fişiere necesare procesului de inserţie şi anume: modelul de limbă şi harta corespondeţelor dintre cuvinte. Căile către aceste fişiere vor fi păstrate într-un fişier text păstrat în cadrul proiectului, iar accesarea lor se face prin intermediul unei clase *Property* care face încarcarea fiecărui fişier din calea specificată. Un exemplu de utilizare a acestei clase este cel pentru încărcarea modelului de limbă:

```
Property.loadProps();
_pathLM = Property.props.getProperty("_pathLM");
```

Variabila se tip String ***_pathLM*** reţine calea către modelul de limbă, deschiderea şi închiderea fişierelor fiind lăsată în seama clasei *Property*. Testul acestei metode de folosire a fişierelor se va face pe computerul personal, de aceea va fi construit încă un document de configurare a căilor pentru serverul care va susţine aplicaţia şi serverul web TomCat, server ce funcţionează cu sistemul de operare Linux.

4.3 SISTEMUL DE RETAURARE A DIACRITICELOR

În acest moment, ar fi o ocazie bună să putem detalia programul de dezambiguizare care stă la baza procedurii de inserţie a diacriticelor. Acesta ia decizia alegerii cuvântului care maximizează produsul probabilităţilor provenite din modelul de limbă şi din harta cuvintelor. Sistemul asupra căruia s-a luat decizia că va fi folosit pentru procesul de restaurare conţinea un model de limbă şi o hartă a cuvintelor formate cu ajutorul utilitarului SRI-LM. Corpusurile de text care au fost folosite pentru crearea acestor rezurse reprezintă o colecţie de discuţii europarlamentare, articole de ziare şi literatură. Unul din aceste corpusuri a fost numit europarl, de dimensiuni 225k propoziţii şi 5.3M cuvinte şi celălalt este o reuniune de texte din literatură şi articole de ziare de dimensiuni 400k propoziţii şi 11M cuvinte. Din aceste două a rezultate un model de limbă de 32M cuvinte (împreună cu probabilităţile lor [2,5GB]) şi o hartă a cuvintelor 4M cuvinte [40MB]. Acest sistem care foloseşte modelele de limbă bazate pe 5-grame, şi care este cel mai efficient de la ora actuală, oferă un WER de 1,99% şi un ChER de 0,48%. Sistemul avea o performanţă ridicată, dar necesită şi un timp de prelucrare îndelungat.

Pentru testarea serviciului web, pe tot parcursul implementării sale a fost folosit un model de limbă de ordin 3, de dimensiuni mai mici (473kB). Acest model de limbă s-a format cu aplicaţia SRI-LM din 30 de articole din ziarul „9-am”, disponibil pe Internet. Această abordare a fost neceară pentru ca aplicaţia să ruleze în timp real. Pentru aplicaţiile care vor folosi sistemul, se va utiliza modelul de limbă extins. Harta cuvintelor a rămas aceeaşi. Rezultatele pe care le oferă aplicaţia nu sunt importante deoarece aceasta a dost creată doar cu scop demonstativ, făcându-se astfel un compromis între viteză şi precizie.

În continuare se va descrie modalitatea de creare a modelului de limbă şi a hărţii cuvintelor. Deşi aceste articole concatenate conţin diacritice, asupra lor a fost aplicat serviciul de preprocesare a textului oferit de pachetul javaNLP2. Programul java, aşa cum am discutat într-un capitol

anterior a uniformizat textul și i-a oferit un aspect monoton prin aplicarea următoarelor proceduri:

- Uniformizarea diacriticelor și a cratimelor;
- Înlocuirea URL cu formele lor vorbite;
- Înlocuirea adreselor de e-mail cu formele lor vorbite;
- Înlocuirea abrevierilor și a numerelor cu formele lor extinse;
- Înlocuirea textului dintre paranteze pe linii diferite și ștergerea parantezelor;

Textul curățat conține 13271 cuvinte și 750 de linii. Un fragment din aceste articole concatenate și curățate ar fi mai concludent:

```
mutu va juca iar la dinamo
unu tipăreşte acest articol
cred că a ales dinamo pentru că a petrecut o perioadă frumoasă aici unde
a câştigat şi campionatul ioan andone antrenor dinamo adrian mutu a rupt
contractul de impresariere cu firma fraţilor becali şi a lui gică popescu
el îşi va continua cariera la dinamo echipa de la care a plecat în iarna
lui două mii spre fotbalul mare
astăzi de la ora doisprezece şi zero minute
```

Cu ajutorul acestui text s-a creat modelul de limbă de tip trigrame. Acest model ar fi de fapt un fișier text care conține trei cuvinte pe fiecare rând, însă datorită utilizării metodelor de întoarcere prezentate într-un capitol anterior (metodă care se bazează pe ideea că dacă o succesiune de trei cuvinte nu a apărut în modelul de limbă, ceea ce este un caz foarte probabil, atunci sistemul „va face un pas înapoi” și va căuta în modelul de limbă format din bigrame și dacă nu găsește nicio potrivire atunci va lua ca referință un model de limbă care nu se folosește de vreo istorie a cuvintelor, și anume, cel care se bazează pe unigrame). Pe lângă succesiunile de cuvinte separate pe linie nouă, în acel fișier sunt trecute și probabilitățile lor de apariție.

```
0.9106374   cele mai mari
0.526639    cumpără mai mult
0.7027303   mult mai vizibili
0.827669    nici mai mult
0.827669    nici mai puţin
0.526639    sută mai mult
0.526639    la manifestaţie au
```

Se poate aprecia ușor că fișierul cu modelul de limbă este mai amplu pentru că are în componență toate unigramele, bigramele și trigramele, ajungând la 14272 linii și 44179 cuvinte.

Harta cuvintelor este și ea un element important în mecanismul de restaurare:

```
intrajutorare intrajutorare 0.019607844 întrajutorare 0.98039216
intram intram 0.05590584 întram 2.1017234E-4 intrăm 0.9409416 întrăm
0.0029424129
intramolecularule intramolecularule 1.0
intransigenta intransigenta 0.0066225166 intransigența 0.37086093
intransigentă 0.3178808 intransigență 0.30463576
```

Se poate observa că primul cuvânt de pe prima linie este un cuvânt fără diacritice și este, de asemenea, un cuvânt ambiguu, pentru că forma sa a fost întâlnită, în textul pe care se bazează această hartă, în două feluri diferite: intrajutorare și întrajutorare, numai că prima forma nici nu este corectă ci constituie o eroare de redactare, de aceea și probabilitatea sa de apariție este foarte mică: 2%. Următorul exemplu este și mai concludent, forma lui „intram” poate să exprime imperfectul verbului „a intra” la persoana I, plural, sau poate să fie forma sa de perfect simplu: „intrăm”, care este și mai folosită, având o probabilitate de 94%. Celelalte două forme cărora le sunt atribuite valori foarte mici reprezintă, din nou, greșeli de redactare. Cazul cel mai

simplu este când nu există nicio ambiguitate între cuvinte și atunci forma lor corectă din limba română este una singură (100% probabilitate).

Astfel, conform formulei următoare se poate calcula probabilitatea de apariție a fiecărui cuvânt din textul de intrare:

$$\hat{w} = \arg_{w_k} \max p(w_k|w_i, w_j) \times p(w_k|w'), \text{ unde}$$

$$p(w_k\ |w_i, w_j) = \frac{Număr\ apariţii\ (w_i, w_j, w_k)}{\sum_w Număr\ apariţii\ \ (w_i, w_j, w)}$$

Prima probabilitate din ecuația întâi este dată de modelul de limbă format din trigrame, în timp ce cealaltă este dată de harta cuvintelor cu corespondențe una-la-mai-multe, iar w' reprezintă cuvântul ambiguu care trebuie rezolvat.

4.4 ARHITECTURA SISTEMULUI

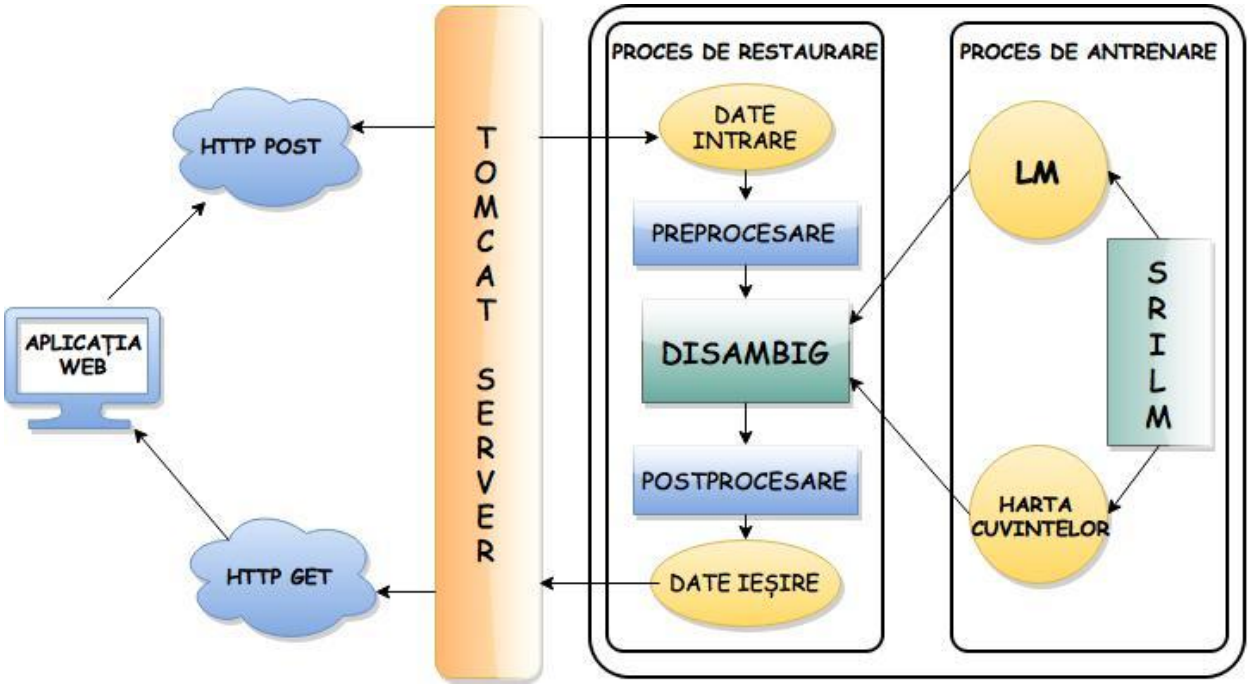

Fig. 3 Arhitectura Sistemului de Restaurare a Diacriticelor

Artectura sistemul se bazează pe o serie de procese consecutive la care este supus un text oareacare introdus direct de la tastatură sau prin încărcarea de pe calculatorul personal. Odată facută cererea de restaurare a diacriticelor aceasta este de fapt o cerere http de tip POST pentru că se trimit date către serverul de aplicație iar apoi prin intermediul unei cereri de tip GET este adus răspunsul sistemul și introdus la locul potrivit în pagina web. Procesarea în limbaj java se realizează în mai multe etape, așa cum se poate observa și în Fig. 3.

Procesul disambig este chemat spre a-și executa rutina printr-un script în linie de comandă Linux. Limbajul java, fiind foarte flexibil și cu multe funcționalități permite execuția unor astfel de comenzi. Se va crea un obiect static Runtime, ce face parte din clasa java.lang.object. Acestă entitate java permite lansarea în execuția a unor comenzi care sunt valabile doar pentru mediul de dezvoltarea pe care este instalată aplicația java. Un exemplu de copiere a conținutului unui fișier în altul este următorul:

```
String[] cmdArray = {"bash", "-c", "cp "+_pathoutput+" "+ _pathcorpus};
Process runCmd = Runtime.getRuntime().exec(cmdArray);
```

Mai întâi se realizează un vector de caractere care reprezintă comanda propriu-zisă, iar apoi se realizează un proces în bash.

Programul de inserție a diacriticelor este disponibil și pe Windows și pe Linux, dar momentan implementarea sistemul se realizează în Linux, mai ales că sistemul de operare al serverului de aplicații este Linux.

Scripturile bash sunt caracterizate de comenzi care nu necesită o compilare în prealabil, dacă o comandă nu este scrisă corect, pur și simplu, nu returnează nimic sau returnează doar un mesaj de eroare. Bash este un procesor de comenzi, care rulează de obicei în ferestre de text, unde utilizatorul scrie comenzi care determină anumite acțiuni. Bash citește, de asemenea, comenzi din fișierul care se numește script și care poate ajunge la o complexitate crescută prin introducerea instrucțiunilor condiționale, a buclelor, sau a funcțiilor speciale. Se permite și procesarea unor parametrii de intrare și lucrul cu variabile. Bash interacționează direct cu sistemul de operare.

Comanda linux care realizează procesul de dezambiguizare este următoarea:

```
String[] cmdArray3 = {"bash", "-c", "/usr/local/srilm/bin/i686-
m64/disambig -lm" + _pathLM + "-order 3 -map " + _pathwordMap + " -text
"+_pathcorpusPreprocessed+" -keep-unk -fb | sed -e 's/<s> //g; s/
<\\/s>//g' > "+_pathcorpusRsd};
Process runCmd3 = Runtime.getRuntime().exec(cmdArray3);
```

După ce semnalează în corpul comenzii că este vorba de o comandă în bash, se va preciza calea către programul de dezambiguizare. Acest program a fost instalat și pe computerul personal și pe serverul pe care rulează aplicația. Calea precizată aici este cea de pe server. Fanionul „-lm” semnalează că următorul parametru introdus este calea către modelul de limbă, iar „-order” setează ca la baza modelului de limbă să fie trigramele. Este o comandă linux destul de amplă, fiind necesară definirea mai multor parametrii de intrare. Următorul fanion ”-map” semnalează faptul că se va încărca și harta cuvintelor, apoi nu în ultimul rând prin „-text” se introduce în procesul de dezambiguizare textul uniformizat. Opțiunea „keep-unk” face ca acele cuvinte care nu există în hartă să nu fie marcate cu eticheta <unk>, cum s-ar face în mod implicit, ci nu vor fi procesate și vor rămâne așa cum sunt. Ultimul fanion „-fb” precizează că în cazul în care nu au fost găsite anumite cuvinte în modelul de limbă se va aplica metoda de întorcere la un model de limbă cu mai puține n-grame. Asupra acestei comenzi se mai aplică un filtru care va șterge toate cuvintele <s> care există în textul restaurat. Prin intermediul semnului „>” se va introduce rezultatul într-un fișier, iar calea către fișier este precizată în valoarea parametrului final.

În concluzie, după ce programul disambig și-a terminat execuția urmează postprocesarea datelor care va readuce textul la forma sa finală.

O demonstrație a funcționării sistemului se poate observa în imaginile de mai jos. În prima dintre ele se poate observa pagina de start a aplicației web, iar în cea de-a doua a fost încărcat un text oarecare pentru și a fost primit și răspunsul sistemului:

Serviciu Web de Restaurare a Diacriticelor

Upload Submit Save As

Serviciu Web de Restaurare a Diacriticelor

Desi vasal al regelui Ungariei, asa cum este numit in mai multe documente ale vremii, Basarab daduse destule semne de rebeliune fata de cel ce ar fi trebuit sa ii fie suveran… Carol Robert de Anjou. In anul 1323, Basarab il sprijina militar pe tarul bulgar Mihail Sisman in lupta impotriva Imperiul Bizantin, o actiune ce incalca vasalitatea sa fata de regele Ungariei. Sapte ani mai tarziu, la 28 iulie 1330, voievodul muntean este alaturi de acelasi Mihai Sisman in lupta de la Velbuzd, impotriva regelui sarb Stefan Decanski, un aliat al lui Carol Robert de Anjou. Probabil, acesta a si fost motivul pentru care regele ungur decide sa dea o lectie rebelului valah si sa il indeparteze de la domnie. La doar cateva saptamani de la acest ultim eveniment, Carol da ordinul de adunare a armatei celei mari a Ungariei. Cu

Deși vasal al regelui Ungariei, așa cum este numit în mai multe documente ale vremii, Basarab dăduse destule semne de rebeliune față de cel ce ar fi trebuit să îi fie suveran¦ Carol Robert de Anjou. În anul 1323, Basarab îl sprijină militar pe țarul bulgar Mihail Șișman în lupta împotriva Imperiul Bizantin, o acțiune ce încalcă vasalitatea să față de regele Ungariei. Șapte ani mai târziu, la 28 iulie 1330, voievodul muntean este alături de același Mihai Șișman în lupta de la Velbuzd, împotriva regelui sârb Ștefan Decanski, un aliat al lui Carol Robert de Anjou. Probabil, acesta a și fost motivul pentru care regele ungur decide să dea o lecție rebelului valah și să îl îndepărteze de la domnie. La doar câteva săptămâni de la acest ultim eveniment, Carol da ordinul de adunare a armatei celei mari a Ungariei. Cu

Upload Submit Save As

4.5 ERORILE PROCESULUI DE RESTAURARE

Se pot observa diverse erori la prelucrarea datelor, „da” apare în loc de „dă” și, de asemenea cea mai supărătore eroare constă în faptul că un text care conține caractere cu diacritice dar care nu sunt recunoscute de către sistem în momentul când se face achiziția datelor, aceste caractere vor fi transformate într-o grupare de caractere necunoscute și dacă, înainte de inserția diacriticelor textul era lizibil, după acest proces nu se mai pune această problemă, textul este aproape indescifrabil. Un astfel de exemplu este încercarea următoare:

„Bătălia de la Posada este numele unui conflict militar între Regatul Ungariei și Țara Românească, petrecut în toamna anului 1330. Această bătălie a marcat emanciparea Țării Românești de sub tutela coroanei maghiare.”

Rezultatul:

„Bă□ tă□ lia de la Posada este numele unui conflict militar Â®ntre Regatul Ungariei è□ i è□ ara RomÂ¢neascä□ , petrecut Â®n toamnă anului 1330. Această□ bă□ tă□ lie a marcat emanciparea è□ ä□ rii RomÂ¢neè□ ti de sub tutela coroanei maghiarè.

Soluțiile acestor erori vor fi implementate într-o altă versiune a sistemului printr-o simplă cercetare a caracterelor care substituie, în general, pe cele cu diacritice. După această etapă, se va aplica o înlocuire a fiecărui caracter cu cel corespunzător și abia după acest pas va fi realizată și preprocesarea datelor.

CAPITOLUL 5

CONCLUZII

5.1 CONCLUZII GENERALE

Metodele de restaurare a diacriticelor constituie o sarcină extrem de utilă, mai ales că la momentul actual multe documente sunt redactate fără diacritice, iar pentru documentele oficiale lipsa diacriticelor nu este nici măcar tolerată. Pentru cei cărora le este greu să redacteze corect un text, şi pentru că se mai pot strecura erori, dar şi pentru textele vechi care sunt păstrate fără diacritice, o metodă de restaurare automată este bine-venită.

În alegerea metodei de inserţie trebuie să se studieze resursele care sunt necesare pentru un astfel de proces. Dintre cele trei metode enunţate în acest proiect cel care nu are nevoie de resurse importante, ci doar de un text curat şi corect este metoda care foloseşte modele de limbă la nivel de caracter. Se realizează o istorie a caracterelor precedente, şi în funcţie de această istorie se poate anticipa caracterul următor. Altă metodă de inserţie a diacriticelor este, şi cea care este folosită în sistemul descris anterior şi necesită un utilitar special de dezambiguizare şi încă două resurse, realizate şi ele printr-o procesare anterioară a unor texte ample (pentru o precizie mai bună). Este vorba de un model de limbă care se bazează pe n-grame. N-gramele reprezintă o succesiune de cuvinte care apar în text cu o anumită probabilitate, iar cealaltă resursă este o hartă a cuvintelor: pentru fiecare cuvânt din textul de antrenare s-au scos diacriticele şi pe fiecare rând al acestui fişier (*wordMap*) este trecut cuvântul lipsit de diacritice, apoi fiecare tipar de cuvânt care ar putea constitui un cuvânt valid în limba română, împreună cu probabilitatea sa de apariţie. O altă metodă de restaurare şi cea mai complexă este cea bazată pe algoritmi de

etichetare. Fiecărui cuvânt din text este analizat sintactic, pentru a elimina ambiguitatea. Dacă totuși rămâne o anumită ambiguitate, ca în cazul „fața” și „fâța”, în care amândouă sunt substantive comune articulate, se realizează o inserție a diacriticelor la nivel de caracter. Un program de marcare nu este ușor de realizat, de aceea această ultimă metodă este mai dificil de implementat.

După alegerea metodei de inserție, pentru contruirea serviciului web accesibil tuturor utilizatorilor a fost nevoie de o alegere a mediilor de dezvoltare și a tehnologiilor folosite. Pentru a testa funcționalitatea programului de inserție automată s-a creat o aplicație web cu ajutorul paginilor JSP. Apoi, pentru realiza o standardizare a cererilor de restaurare s-a folosit protocolul REST care conține metode specifice de trimitere si de solicitare a informațiilor către și de la server. Pentru toate fișierele necesare procesării textului de intrare s-a creat o clasă java care să realizeze citirea documentelor. Metodele principale sunt scrise în limbaj java și realizează preprocesarea textului, transmite comezi de tip bash către sistemul de operare existent pe server, iar, în urma postprocesării, textul este întors către client.

5.2 CONTRIBUȚII PERSONALE

Contribuțiile personale ale autorului se găsesc în Introducere și în Cap. IV, unde se descriu mai întâi ceea ce s-a dorit de la sistem, și respectiv cum au fost acestea implementate și care sunt rezultatele. Sistemul de pre și post-procesare a fost un punct de start și după rezolvarea erorilor care au apărut în sistem, a fost realizat o aplicație web prin intermediul paginilor JSP. Această abordare a fost urmărită datorită faptului că utilizarea paginilor JSP este foarte rapidă și ușor de înțeles. Codul java a fost inserat în interiorul tag-urilor HTML, și a fost apelată o clasă care să realizeze automatizarea procesului de inserție a diacriticelor.

Prin folosirea REST api s-a realizat o cunoștere în detaliu a serverului Tomcat Apache care va lansa în execuție aplicația de oriunde în internet. S-au înțeles și folosit cele mai importante metode ale protocolului REST, adică POST și GET. Pentru buna funcționare și comunicație cu containerul Servlet-ului s-a redactat și descriptorul *web.xml,* ce conține particula URL-ului și extensia acestuia, precum și menționarea obiectului care face cererea de tip POST către server, și anume butonul Submit.

O cercetare mai amănunțită a fost necesară și pentru introducerea scripurilor bash în codul java. Această abordare a fost imperios necesară pentru a lansa în execuție programul de dezambiguizare.

Pentru stabilirea conexiunii dintre conținutul paginii HTML, adică textul trimis de client, și metoda de procesare a textului s-a folosit cod javaScript. În acest cod cel mai important aspect este constituirea URL-ului, deschiderea conexiunii și trimiterea variabilei text de intrare împreună cu valoarea sa către sistemul de restaurare. Rezultatul va fi returnat de o funcție înapoi către client.

Modelul de limbă și harta cuvintelor au fost disponibile pe tot parcursul lucrului la proiect, studiul acestora a fost însă necesar pentru a înțelege funcționarea programului disambig.

5.3 ACTIVITATE ULTERIOARĂ

Așa cum s-a menționat și în introducere, serviciul web primește o plus-valoare în momentul în care este integrat în alte sisteme de sine stătătoare, cum ar fi realizarea unui dicționar pentru limba română sau a unei baze de date, alcătuită din multe articole, asupra cărora este necesară

inserția diacriticelor pentru ca o căutare să intoarcă rezultatele așteptate și nu alte texte care nu au nicio legătură cu domeniul, cum ar fi căutarea construcției „peste apă”, să întoarcă și articole care conțin construcția „pește *in* apă”.

În viitor se dorește să fie rezolvată problema diacriticelor care nu sunt recunoscute de sistem și ale căror coduri ASCII duc la reprezentări dintre cele mai bizare.

Trebuie rezolvată problema formatului textelor, o abordare destul de complicată, pentru că ar presupune o căutare după următoarele aspecte într-un fișier text:

- Antete, titluri, subcapitole care trebuie aduse la un aspect uniform. Este nevoie să se păstreze într-o memorie dimensiunile și numărul rândurilor goale ca delimitează aceste stiluri ale textelor;
- Formulele trebuie și ele scoase și păstrate în fișiere separate;
- Tabelele, figurile să fie eliminate în procesul de presprocesare;

Acestea fiind realizate procesul își poate urma cursul firesc, iar apoi metoda de postprocesare a textului să poată să reintroducă la locul potrivit aceste aspecte problematice.

Un alt aspect care trebuie abordat este formatul în care este salvat un fișier ce urmează să fie încărcat de pe disk. Acesta este de cele mai multe ori .doc sau .docx. Ar trebui implementată o clasă java special care să facă conversia din acele formate în text simplu și invers.

Un țel care este destul de înalt ar fi realizarea unei metode, în limbaj java, care să realizeze operația de restaurare, deoarece aceasta are o întârziere datorită timpului de încărcare a modelului de limbă și a hărții cuvintelor. O variantă ar putea fi realizarea unui sistem care nu necesită prea multe resurse externe. Pentru implementarea procesul de inserție la nivel de caracter este nevoie de cunoșterea algoritmilor statistici IBL (învățare pe bază de instanțe) și implementarea acestora de la zero. Cea de-a doua abordare ar fi transcrierea metodei realizate de către programul disambig în limbaj java, construindu-se un pachet separate care va fi integrat în sistem și care îl va determina să fie 100% cod java.

Aceste țeluri se doresc a fi împlinite pentru a desăvârși serviciul și pentru a întâmpina nevoile utilizatorilor care sunt din ce în ce mai diverse. Realizarea lor va conduce la apariția unei variante gama a serviciului.

REFERINŢE

Horia Cucu, „*Contribuţii la un sistem de recunoaştere de vorbire continuă, cu vocabular extins, independent de vorbitor pentru limba română*”, pp. 70-78, pp. 25-29, pp. 115-117, 2011

Horia Cucu, Andi Buzo, Laurent Besacier, Corneliu Burileanu „SMT-based ASR domain adaptation methods for under-resourced languages: Application to Romanian”, Speech Communications 56, pp. 200-207, 2014

Lucian Petrică, Horia Cucu, Andi Buzo, and Corneliu Burileanu, „*A Robust Diacritics Restoration System using Unreliable Raw Text Data”, University Politehnica of Bucharest, Speech and Dialogue Research Laboratory, Bucharest, Romania

D. Tufiș și A. Chițu, "Automatic diacritics insertion in Romanian texts", în *Proceedings of the International Conference on Computational Lexicography*, Pecs, Hungary, pp. 185-194, 1999

Rada F. Mihalcea, Vivi A. Năstase „O metodă automtă de restaurare a diacriticelor” în D. Tufiș, Florin Gh. Filip „Limba română în Societatea Informațională – Comunitatea Cunoașterii”, pp. 193-209, 2002

Michael Jaki „Repesentational State Transfer” University of Technology Vienna, pp. 3-15

Dan Tufiș, Alexandru Ceaușu, „DIAC^{+}: A Professional Diacritics Recovering System”, Institute for Artificial Intelligence Romanian Academy

Jorge Cardoso, Amit Sheth (Eds.) „Semantic Web Services and Web Process Composition”, pp. 29-32, 2004

Andreas Stolcke „SRI-LM – An extensible language modeling tookit”, Speech Technology and Research Laboratory

Vivek Chopra, Amit Bakore, Jon Eaves, Ben Galbraith, Sing Li, Chanoch Wiggers „Professional Apache Tomcat 5”, pp.53-54, pp 40

Grigore Lăcriță „Scrierea cu diacritice: indicator al gradului de cultură și profesionalism”„Juridic” http://legestart.ro/scrierea-cu-diacritice-indicator-al-gradului-de-cultura-si-de profesionalism/, 2013

ANEXE

COD JAVASCRIPT – varianta JSP

```
function loadfile(fileid, loadid) {
   document.getElementById(loadid).value = 'Loading...';
   setTimeout(function () {
      loadfile2(fileid, loadid)
   }, 1000);
}

function loadfile2(fileid, loadid) {
   if (!window.FileReader) {
      document.getElementById(loadid).value = 'Your browser does not support HTML5 "FileReader"
function required to open a file.';
   } else {
      fileis = document.getElementById(fileid).files[0];
      var fileredr = new FileReader();
      fileredr.onload = function (fle) {
         var filecont = fle.target.result;
         document.getElementById(loadid).value = filecont;
      }
      fileredr.readAsText(fileis);
   }
}

function savefile(saveid) {
   if (!window.Blob) {
      alert('Your browser does not support HTML5 "Blob" function required to save a file.');
   } else {
      var txtwrt = document.getElementById(saveid).value;
      txtwrt = txtwrt.replace(/\n/g, '\r\n');
      var textblob = new Blob([txtwrt], {type: 'text/plain'});
      var saveas = "output.txt";
      var dwnlnk = document.createElement('a');
      dwnlnk.download = saveas;
      dwnlnk.href = window.webkitURL.createObjectURL(textblob);
      dwnlnk.click();
   }
}
```

COD JSP – Pagină de start

```
<%@page import="java.io.OutputStream"%>
<%@page import="java.io.FileOutputStream"%>
<%@page import="java.io.FileReader"%>
<%@page import="java.io.BufferedReader"%>
<%@page import="java.io.File"%>
<%@page import="org.etti.nlp.cleaner.TextCorpusCleaner"%>
<%@page import="org.etti.nlp.diacritics.RestorationCorpusProcessor"%>
<%@page import="Files.MakeFile" %>
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
   <head>
      <script type="text/javascript" src="Functions.js"></script>
      <link rel="stylesheet" type="text/css" href="Style.css" />
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Serviciu Web de Restaurare a Diacriticelor</title>
   </head>
   <body>
      <h1>Serviciu Web de Restaurare a Diacriticelor</h1>
      <br>
      <form action="da.jsp" id="container">
         <textarea id="input" rows="13" cols="68" name="textarea1" form="container" wrap="hard">
<%StringBuffer text = new StringBuffer(request.getParameter("textarea1"));
               out.println(text);
               String str = new String(text);%></textarea> <br>
         <textarea id="output" rows="13" cols="68" name="comment2" wrap="hard"><%
            File file_in = new File("/home/ana/corpus");
            MakeFile restoredFile = new MakeFile(file_in, str);
            restoredFile.process();
            BufferedReader br = new BufferedReader(new FileReader("/home/ana/corpus.out"));
            //out.println(br);
            String line = null;
            while ((line = br.readLine()) != null) {
               out.println(line);
            }
         <br>
         <input type="file" id="file" onChange="setTimeout('loadfile(\'file\',\'input\')', 100)"
style="display:none;"/>
         <input type="button" value="Upload" onclick="document.getElementById('file').click();" />
         <input type="submit" value="Submit">
         <input type="button" value="Save As" onClick="savefile('output');" />


      </form>
   </body>
</html>
```

COD JSP – Pagină de răspuns

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
```

```
<html>
   <head>
      <script type="text/javascript" src="Functions.js"></script>
      <link rel="stylesheet" type="text/css" href="Style.css" />
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Serviciu Web de Restaurare a Diacriticelor</title>
   </head>
   <body>
      <h1>Serviciu Web de Restaurare a Diacriticelor</h1>
      <BR>
      <form action="da.jsp" id="container">
         <textarea id="input" rows="13" cols="68" name="textarea1" form="container"
wrap="hard">Bine ați venit! Aici veți insera textul!
         </textarea> <br>
         <textarea id="output" rows="13" cols="68" name="comment2" wrap="hard">
         </textarea> <br>
         <br>
         <input type="file" id="file" onChange="setTimeout('loadfile(\'file\',\'input\')', 100)"
style="display:none;"/>
         <input type="button" value="Upload" onclick="document.getElementById('file').click();" />
         <input type="submit" value="Submit">
         <input type="button" value="Save As" onClick="savefile('output');" />
      </form>
   </body>
</html>
```

COD JAVA – varianta JSP

```
package Files;

import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStreamWriter;
import java.lang.ProcessBuilder.Redirect;
import java.util.Map;
import java.util.logging.Level;
import java.util.logging.Logger;
import org.etti.nlp.cleaner.TextCorpusCleaner;
import org.etti.nlp.diacritics.RestorationCorpusProcessor;

public class MakeFile {

   public MakeFile(File fout, String s) throws IOException, Exception {

      FileOutputStream fos = new FileOutputStream(fout);

      OutputStreamWriter osw = new OutputStreamWriter(fos);
      osw.write(s);
      osw.close();
   }

  public void process() {
      String file_in = new String("/home/ana/corpus");
```

```
        String file_out = new String("/home/ana/corpus.out");
        String file_rsd = new String("/home/ana/corpus.rsd-preliminary");
        String file_in_pre = new String("/home/ana/corpus.preprocessed");
        String file_in_punct = new String("/home/ana/corpus.punctuation");
        String file_in_case = new String("/home/ana/corpus.case");
        String config = new String("config.prop");
        TextCorpusCleaner textcorpuscleaner = new TextCorpusCleaner(config, file_in, 1, 1, false, false,
false);
        RestorationCorpusProcessor processor = new RestorationCorpusProcessor(true, file_in, file_in_pre,
file_in_punct, file_in_case);
        RestorationCorpusProcessor postproccesor = new RestorationCorpusProcessor(false, file_rsd,
file_out, file_in_punct, file_in_case);

        try {
            String[] cmdArray1 = {"bash", "-c", "sed '/^\\s*$/d' /home/ana/corpus > /home/ana/output"};
            Process runCmd1 = Runtime.getRuntime().exec(cmdArray1);
            runCmd1.waitFor();
            String[] cmdArray2 = {"bash", "-c", "cp /home/ana/output  /home/ana/corpus"};
            Process runCmd2 = Runtime.getRuntime().exec(cmdArray2);
            runCmd2.waitFor();
            processor.process();
            String[] cmdArray3 = {"bash", "-c", "/usr/share/srilm/bin/i686-m64/disambig -lm /home/ana/LMe
-order 3"
            +"-map /home/ana/wordMap -text /home/ana/corpus.preprocessed -keep-unk -fb | sed -e 's/<s>
//g; s/ <\\/s>//g' > /home/ana/corpus.rsd-preliminary"};
            Process runCmd3 = Runtime.getRuntime().exec(cmdArray3);
            runCmd3.waitFor();
            postproccesor.process();
            InputStream Is = p.getInputStream();
            int i = 0;
            StringBuffer sb = new StringBuffer();
            while ((i = Is.read()) != -1) {
               sb.append((char) i);
            }
            System.out.println(sb.toString());
        } catch (IOException e) {
            e.printStackTrace();
        } catch (Exception ex) {
            Logger.getLogger(MakeFile.class.getName()).log(Level.SEVERE, null, ex);
        }
      }
}
```

COD HTML – varianta REST

```
<!DOCTYPE html>
<html>
   <head>
      <script type="text/javascript" src="Scripts.js"></script>
      <link rel="stylesheet" type="text/css" href="Styles.css" />
      <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
      <title>Serviciu Web de Restaurare a Diacriticelor</title>
   </head>
   <body>
      <h1>Serviciu Web de Restaurare a Diacriticelor</h1>
```

```
      <BR>
      <form  name="myform" method="POST" enctype="multipart/form-data" id="info"
onsubmit="return false">
         <textarea id="inputText" rows="13" cols="68" name="textarea1" form="container"
wrap="hard"> </textarea> <br>
         <textarea id="outputText" rows="13" cols="68" name="comment2" wrap="hard">
         </textarea> <br>
         <br>
         <input type="file" id="file" onChange="setTimeout('loadfile(\'file\',\'inputText\')', 100)"
style="display:none;"/>
         <input type="button" value="Upload" onclick="document.getElementById('file').click();" />
         <input type="submit" value="Submit" onclick="mySubmit()">
         <input type="button" value="Save As" onClick="savefile('outputText');" />
      </form>
   </body>
</html>
```

COD JAVASCRIPT – varianta REST

```
function mySubmit(){
//var baselink = "http://localhost:8080/DiacriticsRESTorer/ani/restorer/process";
var baselink = "http://dev.speed.pub.ro:10082/DiacriticsRESTorer/ani/restorer/process";
var inputText;
inputText = document.getElementById("inputText").value;
termination = "?inputText=" + inputText;
var xmlHttp = new XMLHttpRequest();
xmlHttp.open("GET", baselink + termination, false);
xmlHttp.send();
alert("s-a facut trimiterea datelor!");
var responseText = xmlHttp.responseText;
         document.getElementById("outputText").value=responseText;
}

function loadfile(fileid, loadid) {
   document.getElementById(loadid).value = 'Loading...';
   setTimeout(function () {
      loadfile2(fileid, loadid)
   }, 1000);
}

function loadfile2(fileid, loadid) {
   if (!window.FileReader) {
      document.getElementById(loadid).value = 'Your browser does not support HTML5 "FileReader"
function required to open a file.';
   } else {
      fileis = document.getElementById(fileid).files[0];
      var fileredr = new FileReader();
      fileredr.onload = function (fle) {
         var filecont = fle.target.result;
         document.getElementById(loadid).value = filecont;
      }
      fileredr.readAsText(fileis);
   }
```

```
}

function savefile(saveid) {
   if (!window.Blob) {
      alert('Your browser does not support HTML5 "Blob" function required to save a file.');
   } else {
      var txtwrt = document.getElementById(saveid).value;
      txtwrt = txtwrt.replace(/\n/g, '\r\n');
      var textblob = new Blob([txtwrt], {type: 'text/plain'});
      var saveas = "outputText.txt";
      var dwnlnk = document.createElement('a');
      dwnlnk.download = saveas;
         dwnlnk.href = window.webkitURL.createObjectURL(textblob);
      dwnlnk.click();
   }
}
```

COD JAVA - varianta REST

```
package Files;

import java.io.BufferedReader;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStreamWriter;
import java.util.logging.Level;
import java.util.*;
import java.util.logging.Logger;
import org.etti.nlp.cleaner.TextCorpusCleaner;
import org.etti.nlp.diacritics.RestorationCorpusProcessor;
import javax.servlet.http.HttpServletRequest;
import javax.ws.rs.Consumes;
import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;

@Path("/restorer")
public class MakeFile {

   static String _pathLM;
   static String _pathwordMap;
   static String _pathcorpus;
   static String _pathcorpusOut;
   static String _pathcorpusCase;
   static String _pathcorpusPunct;
   static String _pathcorpusRsd;
   static String _pathcorpusPreprocessed;
```

```
static String _pathconfig;
static String _pathoutput;
static {
   try {
      Property.loadProps();
      _pathLM = Property.props.getProperty("_pathLM");
      _pathwordMap = Property.props.getProperty("_pathwordMap");
      _pathcorpus = Property.props.getProperty("_pathcorpus");
      _pathcorpusOut = Property.props.getProperty("_pathcorpusOut");
      _pathcorpusCase = Property.props.getProperty("_pathcorpusCase");
      _pathcorpusPunct = Property.props.getProperty("_pathcorpusPunct");
      _pathcorpusRsd = Property.props.getProperty("_pathcorpusRsd");
      _pathcorpusPreprocessed = Property.props.getProperty("_pathcorpusPreprocessed");
      _pathconfig=Property.props.getProperty("_pathconfig");
      _pathoutput=Property.props.getProperty("_pathoutput");

   } catch (Exception e) {
      e.printStackTrace();
   }
}

public void makeFile(File fout, String s) throws IOException, Exception {
   FileOutputStream fos = new FileOutputStream(fout);
   OutputStreamWriter osw = new OutputStreamWriter(fos);
   osw.write(s);
   osw.close();
}

String readFile(String fileName) throws IOException {
   BufferedReader br = new BufferedReader(new FileReader(fileName));
   try {
      StringBuilder sb = new StringBuilder();
      String line = br.readLine();

      while (line != null) {
         sb.append(line);
         sb.append("\n");
         line = br.readLine();
      }
      return sb.toString();
   } finally {
      br.close();
   }
}
@Path("/process")
@GET
@Produces(MediaType.TEXT_PLAIN)
@Consumes(MediaType.TEXT_PLAIN)
public String process(@Context HttpServletRequest request) throws Exception {
   String inputText;
   File file_input = new File(_pathcorpus);
   inputText = request.getParameter("inputText");
   BufferedReader br = new BufferedReader(new FileReader(_pathcorpusOut));

   makeFile(file_input, inputText);
   TextCorpusCleaner textcorpuscleaner = new TextCorpusCleaner(_pathconfig, _pathcorpus, 1, 1,
```

```
false, false, false);
        RestorationCorpusProcessor processor = new RestorationCorpusProcessor(true, _pathcorpus,
_pathcorpusPreprocessed, _pathcorpusPunct, _pathcorpusCase);
        RestorationCorpusProcessor postproccesor = new RestorationCorpusProcessor(false,
_pathcorpusRsd, _pathcorpusOut, _pathcorpusPunct, _pathcorpusCase);

        try {
            String[] cmdArray1 = {"bash", "-c", "sed '/^\\s*$/d' "+_pathcorpus+" > "+ _pathoutput};
            Process runCmd1 = Runtime.getRuntime().exec(cmdArray1);
            runCmd1.waitFor();
            String[] cmdArray2 = {"bash", "-c", "cp "+_pathoutput+" "+ _pathcorpus};
            Process runCmd2 = Runtime.getRuntime().exec(cmdArray2);
            runCmd2.waitFor();
            processor.process();
            String[] cmdArray3 = {"bash", "-c", "/usr/local/srilm/bin/i686-m64/disambig "
                + "-lm" + _pathLM + "-order 3 -map " + _pathwordMap + " -text
"+_pathcorpusPreprocessed+" -keep-unk -fb | "
                + "sed -e 's/<s> //g; s/ <\\/s>//g' > "+_pathcorpusRsd};
            Process runCmd3 = Runtime.getRuntime().exec(cmdArray3);
            runCmd3.waitFor();
            postproccesor.process();
        } catch (IOException e) {
            e.printStackTrace();
        } catch (Exception ex) {
            Logger.getLogger(MakeFile.class.getName()).log(Level.SEVERE, null, ex);
        }
        String outputString = null;
        System.out.println(outputString);
        outputString = readFile(_pathcorpusOut);
        return outputString;
    }
}
```

COD CSS

```
h1 {font-family: Allura;
    padding: 14;
    word-spacing: 6px; color: #036BFF; font-size: 31px; font-weight: bold; letter-spacing: -1px; line-
height: 1;}
textarea {
    white-space: normal;
    text-align: justify;
    -moz-text-align-last: left;
    text-align-last: left;
    margin: 0;
}
```