ReTeRom/TEPROLIN 1.5: Definirea specificațiilor funcționale și arhitecturale ale platformei integrate și configurabile de prelucrare a textelor

Radu Ion
Institutul de Cercetări pentru Inteligență Artificială „Mihai Drăgănescu”
Academia Română
radu@racai.ro

1. Introducere

Corpusul bimodal pentru limba română adnotat pe multiple niveluri (COBILIRO) este în esență un corpus de text și voce în care fragmentele de text (de obicei fraze) sunt împerecheate cu fișierele (format WAV) care înregistrează rostirea lor de către diverşi vorbitori nativi ai limbii române. Un astfel de corpus este resursa fundamentală pentru învățarea automată a sistemelor de sinteză a vorbirii[1] care produc automat rostirea în limba română a unor fraze arbitrare și a sistemelor de recunoaștere a vorbirii[2] care traduc automat semnalul vocal în text. Aceste sisteme folosesc adnotări ale frazelor corpusului la diverse niveluri (fonetic, morfologic, sintactic și semantic) împreună cu alinieri ale cuvintelor cu semnalul vocal, pentru a învăța automat fie să sintetizeze o voce umană, fie să transcrie automat un semnal vocal.

O platformă de prelucrare a textelor care să ofere prelucrările utile într-un astfel de corpus trebuie să respecte următoarele cerințe:

- Să fie standardizată, astfel încât noi adnotări să poată fi ușor adăugate;
- Să fie modulară, astfel încât programatorul să poată ușor substitui module în lanțurile de prelucrare, module care efectuează anumite operații cu o acuratețe mai bună;

[1] ”Text To Speech” (TTS) în limba engleză
[2] ”Automatic Speech Recognition” (ASR) în limba engleză

- Să fie accesibilă ca serviciu web, astfel încât mai multe echipe să o poată folosi simultan, fără a o instala.

2. Specificații arhitecturale ale platformei de prelucrare a textelor

Pentru că platforma de prelucrare a textelor pentru COBILIRO (dezvoltată în proiectul TEPROLIN) este o colecție eterogenă de aplicații de prelucrare a limbajului natural, scrise în diverse limbaje de programare, și pentru că dorim ca această platformă să fie disponibilă pe orice calculator cu o conexiune la Internet, fără a trebui să fie instalată[3], platforma *va fi oferită utilizatorului sub forma unui serviciu web de tip REST* (Fielding, 2000).

Serviciul web va expune programatorului o serie de funcții care realizează prelucrările oferite de platformă, funcții care pot fi folosite individual sau ca parte a unui lanț de prelucrare. Fiecare funcție va primi parametri de intrare fie de la utilizator, fie de la o altă funcție a platformei și va furniza un rezultat care va putea fi utilizat imediat sau ca un parametru pentru următoarea funcție din lanțul de prelucrare. Dacă utilizatorul nu dorește să utilizeze prelucrarea oferită de o funcție, acesta va putea specifica acest lucru la apelul funcției (printr-un parametru special) urmând ca funcția să prelucreze parametrii de intrare doar în sensul asamblării acestora în formatul rezultatului oferit.

Pentru a putea compune funcțiile PLN implementate de aplicațiile componente ale platformei TEPROLIN, *trebuie să standardizăm parametrii de I/O și rezultatele calculate de funcții*, standardizare care însă va fi transparentă utilizatorului. Vom folosi un format textual și tabular pentru parametrii de I/O și pentru rezultate, pentru a nu pierde timp de procesare cu parsarea unor formate structurate de tip XML. Formatul intern al parametrilor de I/O și al rezultatelor pe care l-am ales este CoNLL-X (Buchholz și Marsi, 2006) care va fi extins cu coloane suplimentare pentru a acomoda rezultatele furnizate de aplicațiile PLN componente. Figura 1 exemplifică o propoziție în limba română în acest format.

[3] Instalarea ar fi anevoioasă pentru că, fiind compusă din aplicații dezvoltate independent, în diverse limbaje de programare interpretate, utilizatorul ar trebui să instaleze separat interpretoarele și bibliotecile de programe necesare, să citească manuale de instalare, etc.

ID	WORD	LEMMA	CTAG	MSD	FEATS	HEAD	DEP	PHD	PDR
1	Scrisul	scris	NSRY	Ncmsry	_	6	nsubj	_	_
2	în	în	S	Spsa	_	3	case	_	_
3	sine	sine	PXA	Px3--a--------s	_	1	amod	_	_
4	era	fi	V3	Vmii3s	_	6	cop	_	_
5	o	un	TSR	Tifsr	_	6	det	_	_
6	treabă	treabă	NSRN	Ncfsrn	_	0	root	_	_
7	ușoară	ușor	ASN	Afpfsrn	_	6	amod	_	_
8	.	.	PERIOD	PERIOD	_	6	punct	_	_

Figura 1: Formatul tabular CoNLL-X al parametrilor de I/O și rezultatelor funcțiilor

Semnificațiile coloanelor din Figura 1 sunt următoarele:

1. ID: este indexul unității lexicale în frază, pornind de la 1;
2. WORD: este forma ocurentă a cuvântului în frază;
3. LEMMA; este forma standard de dicționar pentru forma ocurentă din coloana „WORD”;
4. CTAG: este eticheta morfo-sintactică „redusă” (Tufiș, 2000) a cuvântului din coloana „WORD”, în context;
5. MSD[4]: este eticheta morfo-sintactică „extinsă” a cuvântului din coloana „WORD”, în context, corespunzătoare etichetei CTAG;
6. FEATS: trăsături sintactico-semantice suplimentare ale cuvântului; dacă acestea lipsesc, se folosește caracterul „underscore”;
7. HEAD: este indexul regentului cuvântului din coloana 2 în analiza sintactică cu relații de dependență;
8. DEP: este relația de dependență (Barbu Mititelu et al., 2016) care se stabilește între coloanele „HEAD” și „ID”;

[4] Vezi http://nl.ijs.si/ME/V4/msd/html/msd-ro.html pentru detalii

9. PHD („parent head”): este ID-ul regentului regentului din coloana „HEAD”;
10. PDR („parent dependency relation”): este relația de dependență care se stabilește între coloanele „HEAD” și „PHD”.

În cadrul TEPROLIN *vom defini o serie de lanțuri de prelucrare (funcții compuse)* pe care le considerăm utile pentru proiectele componente ale ReTeRom (TADARAV - Tehnologii pentru adnotarea automată a datelor audio și pentru realizarea interfețelor de recunoaștere automată a vorbirii și SINTERO - Tehnologii de realizare a interfețelor om-mașină pentru sinteza text-vorbire cu expresivitate) cum ar fi de exemplu adnotarea cu etichete morfo-sintactice a unei fraze sau analiza gramaticală cu arbori de dependențe sintactice (pentru care existența adnotării cu etichete morfo-sintactice este o cerință strictă). Va exista de asemenea *un lanț de prelucrare complet* care va include toate tipurile de procesări pe care platforma le poate realiza.

O provocare pe care o întâlnim la realizarea unei platforme de prelucrare a textelor în limba română alcătuită dintr-o colecție eterogenă de aplicații de prelucrare a limbajului natural (PLN) este viteza de execuție a lanțurilor de prelucrare, mai ales a lanțului de prelucrare complet, în condițiile în care *platforma va fi utilizată și pentru prelucrări de text în timp real*. Funcțiile serviciului web sunt „wrapper”-e[5] ale aplicațiilor componente iar lanțul de prelucrare presupune preluarea rezultatelor unei funcții ca parametri de intrare pentru următoarea funcție din lanț. Aplicațiile componente pot fi executate doar în procese separate (fiecare cu interpretorul potrivit sau direct dacă sunt compilate) însă aceste procese au, de obicei, un cost ridicat (în timp de execuție) de pornire (au nevoie de timp pentru a încărca diverse resurse pe le accesează cum ar fi dicționare, modele statistice, etc.) care nu permite pornirea lor la fiecare apel de funcție. Acest lucru implică păstrarea proceselor rezidente în memoria platformei și comunicarea cu acestea cu unul dintre mecanismele IPC[6] cum ar fi de exemplu „named pipes”, o metodă de comunicare inter-proces în care un fișier rezident în memoria RAM are rol de canal de comunicare (uni- sau bi-direcțional) sincronizat accesibil proceselor active (vezi Figura 2). În acest context, *accesul concurent (pe mai multe fire de execuție sau de la mai mulți clienți HTTP) la platforma de prelucrare a textelor trebuie programat în serviciul web* astfel încât procesele aplicațiilor

[5] https://en.wikipedia.org/wiki/Wrapper_function
[6] https://en.wikipedia.org/wiki/Inter-process_communication

componente ale platformei să servească cereri secvențial. Astfel, în funcție de specificațiile serverului pe care găzduim platforma și serviciul web (număr de nuclee, memorie RAM) putem porni mai multe procese de același tip și servi cereri în ordinea în care apar (cererile sunt plasate într-o coadă de așteptare).

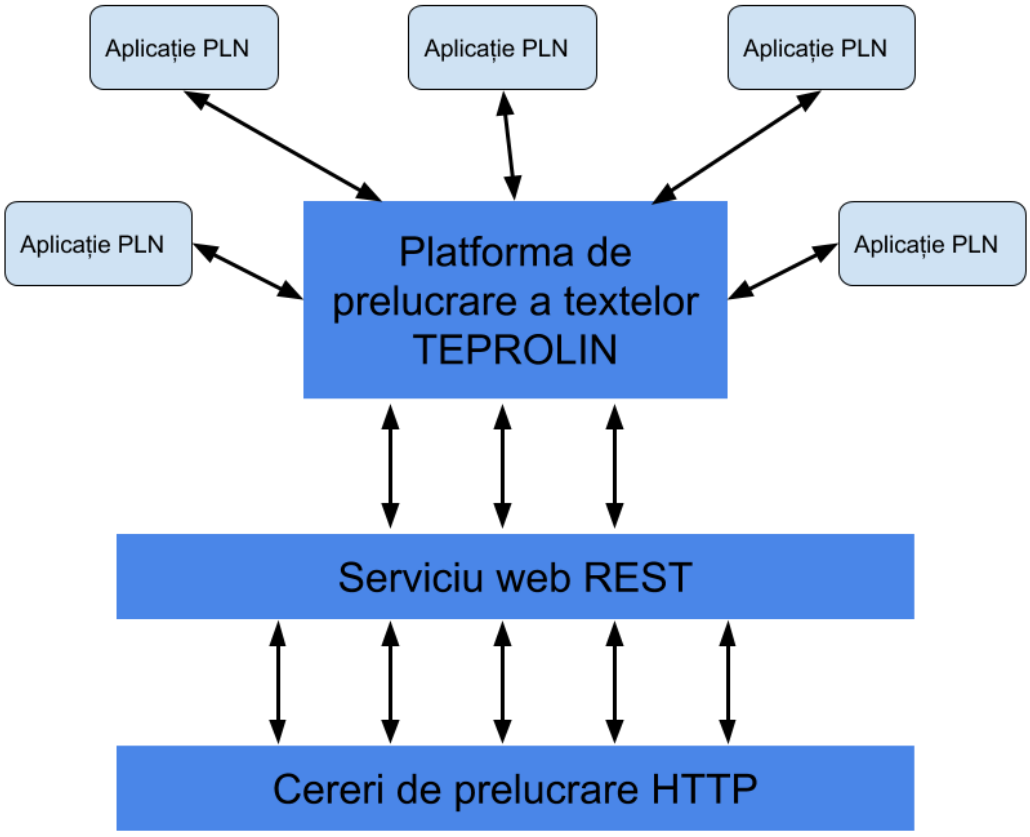

Figura 2: Schema bloc a platformei de prelucrare a textelor TEPROLIN

Rezumând cele expuse mai sus, platforma de prelucrare a textelor TEPROLIN:

1. Este un serviciu web REST cu suport pentru acces concurent pentru un număr fix de clienți (cu atât mai mare cu cât resursele serverului care găzduiește serviciul web permit);
2. Rulează aplicații PLN în procese rezidente în memorie, minimizând timpul de execuție al lanțurilor de prelucrare prin utilizarea mecanismelor IPC;
3. Definește lanțuri de prelucrare a textelor formate din compunerea funcțiilor dar permite utilizatorului și apelarea de funcții individuale prin standardizarea parametrilor de I/O.

4. Rezultatele lanțurilor de prelucrare *vor fi oferite utilizatorului în formatul stabilit de* ***Activitatea 1.1 - Studiu state-of-the-art asupra realizării corpusurilor bimodale***, din cadrul proiectului component COBILIRO.

3. Specificații funcționale ale platformei de prelucrare a textelor

Platforma de prelucrare a textelor TEPROLIN va reuni o mare parte a aplicațiilor de PLN care sunt deja disponibile (și funcționale), dezvoltate de partenerii proiectului ReTeRom. Provocarea proiectului TEPROLIN este armonizarea tuturor acestor aplicații în platforma pe care am descris-o în secțiunea anterioară, printr-o soluție ușor extensibilă care să accepte module noi PLN și definiții ale unor noi lanțuri de prelucrare. Inventarul modulelor PLN existente care vor fi integrate în platforma TEPROLIN și modificările pe care le operăm pe fiecare modul pentru platforma TEPROLIN fac obiectul *Activității 1.6 - Definirea modulelor software și a serviciilor oferite de proiect; identificarea adaptărilor necesare pentru modulele PLN existente și a modulelor noi necesare platformei integrate și configurabile de prelucrare a textelor*.

Platforma TEPROLIN va include următoarele tipuri de preprocesări textuale atomice (nu neapărat în această ordine):

1. **Normalizarea textelor în limba română**: acest modul presupune identificarea diacriticelor românești non-standard (anume „ş” și „ţ”) și înlocuirea lor automată cu variantele standard (cele cu virgulă, anume „ș” și „ț”), normalizarea cratimei (înlocuirea tuturor variantelor Unicode ale cratimei cu cratima standard „-”), normalizarea spațiilor (înlocuirea tuturor variantelor Unicode ale cratimei cu caracterul spațiu standard „ ”), și alte tipuri de normalizări;
2. **Inserarea automată a diacriticelor românești în texte**: acest modul va identifica cuvintele care nu conțin diacritice într-o frază și *va insera automat variantele acestora cu diacritice, în context*. De exemplu, cuvântul „fata” poate fi substantiv comun, feminin, articulat sau nearticulat, caz în care forma corectă este „fată”. În cazul în care este verb, forma cu diacritice a cuvântului poate fi „făta”, „fătă” sau „fată”;

3. **Despărțirea în silabe a cuvintelor**: acest modul va despărți în silabe cuvintele unei fraze și va atașa despărțirea în silabe fiecărei unități lexicale a frazei. De exemplu pentru „camion”, despărțirea în silabe este „ca-mi-on”;
4. **Poziționarea accentului**: acest modul va identifica silaba care poartă accentul cuvântului. De exemplu, pentru „camion”, accentul cade pe ultima silabă a cuvântului „ca-mi-*ón*”;
5. **Transcriere fonetică**: acest modul va „traduce” automat din forma scrisă a cuvântului în reprezentarea fonetică a sa: pentru „doagă”, transcrierea fonetică este [dŏa̲gă] în care accentul este subliniat iar „ŏ” este simbolul pentru semivocala „o”;
6. **Expandarea numerelor în termeni numerali**: acest modul va transcrie automat orice numeral scris în cifre în echivalentul său literal: „124” va fi transcris în ”o sută douăzeci și patru”;
7. **Expandarea abrevierilor**: acest modul va transcrie automat, în context, o serie de abrevieri în formele lor extinse: „DNA” va fi transcrisă în „Direcția Națională Anticorupție”, în contextul în care are acest sens;
8. **Segmentare la nivel de frază**: acest modul va despărți un text dat (paragraf sau document) în frazele componente;
9. **Segmentare la nivel de unitate lexicală**: acest modul va identifica toate cuvintele și semnele de punctuație dintr-o frază și va oferi lista (ordonată) în care acestea apar în frază;
10. **Adnotare cu etichete morfo-sintactice („POS tagging”)**: acest modul va face analiza morfo-sintactică, în context, a fiecărei unități lexicale a frazei. În urma acestei analize, fiecare unitate lexicală a frazei va fi automat etichetată cu una din etichetele MSD[7], aplicabilă în context;
11. **Lematizare**: acest modul identifică forma standard de dicționar a cuvintelor frazei. De exemplu, „deș̲i” poate fi conjuncție (am subliniat accentul), caz în care are lema „deș̲i”, sau poate fi adjectiv la nominativ/acuzativ, plural („deș̲i”), caz în care are lema „des”;
12. **Identificarea constituenților sintactici („Chunking”)**: acest modul detectează limitele (indecșii cuvintelor care le alcătuiesc) grupurilor nominale („o floare albastră”, „mașina

[7] Vezi http://nl.ijs.si/ME/V4/msd/html/msd-ro.html pentru detalii.

de cusut”), verbale („au fost făcute”, „se vor fi dus”), adjectivale („foarte frumoasă”) și adverbiale („destul de repede”) în frază;

13. **Analiza sintactică cu relații de dependență („Dependency parsing”)**: acest modul construiește automat un arbore de analiză sintactică a frazei în care se precizează subiectul propoziției, complementele predicatului, etc.

Operațiile atomice de mai sus pot fi definite ca niște metode Java cu următorii parametri:

1. `String` **`textNormalization`**`(String inputText);` -- aplică operația definită la punctul 1 anterior pe parametrul de intrare `inputText` (un șir de caractere Unicode) și returnează textul normalizat procesărilor ulterioare.
2. `String` **`diacriticsInsertion`**`(String inputText);` -- aplică operația definită la punctul 2 de mai sus parametrului `inputText`. Acest parametru poate fi rezultatul operației `textNormalization`.
3. `List<String>` **`sentenceSplitter`**`(String inputText, boolean goThrough);` -- segmentează parametrul `inputText` într-o listă de fraze (vezi operația 8 definită mai sus), unde `inputText` poate fi mai întâi prelucrat cu lanțul de prelucrare `textNormalization/diacriticsInsertion`. În cazul în care `goThrough` este `true`, `inputText` va fi segmentat după secvența de caractere „end-of-line” (descrisă de expresia regulată „`\r?\n`”).
4. `String` **`sentenceTokenizer`**`(List<String> inputSentences, boolean goThrough);` -- va folosi rezultatul operației `sentenceSplitter` și va segmenta fiecare frază din lista `inputSentences` în unitățile lexicale componente (vezi operația atomică 9 de mai sus). Rezultatul este întors utilizatorului sub forma unui obiect `String` în formatul CoNLL-X în care coloanele 1 și 2 sunt populate cu ID-urile unităților lexicale și cu formele unităților lexicale iar frazele sunt separate de câte o linie vidă. În cazul în care parametrul `goThrough` este `true`, fiecare frază este segmentată utilizând spațiul ca separator.
5. `String` **`syllabifyWords`**`(String conllxText, boolean goThrough);` -- va despărți în silabe (vezi operația atomică 3 de mai sus) cuvintele din parametrul `conllxText`, după ce acesta este prelucrat cu `sentenceTokenizer`. Rezultatul va fi depus într-o nouă coloană a formatului CoNLL-X și întors utilizatorului. Dacă parametrul `goThrough` este

`true`, coloana corespunzătoare cuvintelor despărțite în silabe va fi vidă (populată cu „_” pentru fiecare cuvânt).

6. `String` **`posTagger`**`(String conllxText, boolean goThrough);` -- va adnota cu etichete morfo-sintactice (operația atomică 10 de mai sus) fiecare unitate lexicală a parametrului `conllxText` (care trebuie să fie prelucrat cu `sentenceTokenizer` în prealabil) și va returna obiectul `String` în formatul CoNLL-X în care coloanele 4 și 5 vor fi populate cu etichetele corespunzătoare. Dacă parametrul `goThrough` este `true`, coloanele 4 și 5 vor fi vide (populate cu „_” pentru fiecare cuvânt).
7. `String` **`wordsStressIdentification`**`(String conllxText, boolean goThrough);` -- va marca accentul pe silaba corespuzătoare a fiecărui cuvânt din parametrul `conllxText`. Acest parametru trebuie să fie procesat în prealabil cu operația `posTagger`. Rezultatul va fi depus pe o nouă coloană a formatului CoNLL-X și întors utilizatorului.
8. `String` **`phoneticTranscription`**`(String conllxText, boolean goThrough);` -- va transcrie fonetic fiecare cuvânt din parametrul `conllxText`. Acest parametru trebuie să fie procesat în prealabil cu operația `sentenceTokenizer`. Rezultatul va fi depus pe o nouă coloană a formatului CoNLL-X și întors utilizatorului.
9. `String` **`expandNumerals`**`(String conllxText, boolean goThrough);` -- va transcrie fiecare numeral scris cu cifre din parametrul `conllxText` în echivalentul său literal (vezi operația 6 de mai sus). Acest parametru va fi procesat în prealabil cu operația `posTagger`. Rezultatul va fi depus pe o nouă coloană a formatului CoNLL-X și întors utilizatorului.
10. `String` **`expandAbbreviations`**`(String conllxText, boolean goThrough);` -- va găsi formele complete pentru fiecare abreviere din parametrul `conllxText` (vezi operația 7 de mai sus). Acest parametru va fi procesat în prealabil cu operația `posTagger`. Rezultatul va fi depus pe o nouă coloană a formatului CoNLL-X și întors utilizatorului.
11. `String` **`lemmatizeWords`**`(String conllxText, boolean goThrough);` -- va lematiza fiecare cuvânt din parametrul `conllxText`. Acest parametru trebuie să fie procesat în prealabil cu operația `posTagger`. Rezultatul va fi depus pe coloana 3 a formatului CoNLL-X și întors utilizatorului.
12. `String` **`chunker`**`(String conllxText, boolean goThrough);` -- va identifica constituenții sintactici din parametrul `conllxText`. Acest parametru trebuie să fie

procesat în prealabil cu operația `posTagger`. Rezultatul va fi depus pe coloana 6 a formatului CoNLL-X, alături de alte trăsături, și întors utilizatorului.

13. `String` **`dependencyParser`**`(String conllxText, boolean goThrough);` -- va identifica relațiile de dependență din parametrul `conllxText`. Acest parametru trebuie să fie procesat în prealabil cu operațiile `posTagger`, `lemmatizeWords` și `chunker`. Rezultatul va fi depus pe coloanele 7 și 8 ale formatului CoNLL-X și întors utilizatorului.

Cu aceste definiții, putem da următoarele exemple de lanțuri de prelucrare în platforma TEPROLIN:

`diacriticsInsertion(textNormalization(”o șarada”))` -- în care cuvântul „șarada” este întâi normalizat la „șarada” iar apoi, algoritmul de inserție de diacritice va detecta formularea corectă „o șaradă”.

`sentenceTokenizer(sentenceSplitter(diacriticsInsertion(”A fost odata ca niciodata.”)))` -- în care, după efectuarea tuturor operațiilor din stiva de apel, vom avea segmentarea în unități lexicale „A”, „fost”, „odată”, „ca”, „niciodată”, „.”, în format CoNLL-X.

Lanțul lexical complet se va face prin apelarea înlănțuită a tuturor operațiilor de mai sus, de la operația nr. 13 la operația nr. 1.

4. Concluzii

Prezentul raport descrie arhitectura platformei de prelucrare a textelor TEPROLIN și modul general în care modulele PLN pot fi interconectate în lanțuri de prelucrare complexe, cu timpi de execuție reduși. Am identificat și definit (în jargon Java) 13 operații elementare de prelucrare a textelor și cerințele de interconectare a acestora în ce privește parametrii de I/O.

TEPROLIN va fi așadar un serviciu web REST care va conține interfețe Java de implementare a operațiilor identificate, putând fi utilizat însă și ca un pachet în limbajul de programare Java.

5. Referințe bibliografice

Barbu Mititelu, Verginica, Ion, Radu, Simionescu, Radu, Irimia, Elena and Perez, Cenel-Augusto (2016). The Romanian Treebank Annotated According to Universal Dependencies. In Proceedings of HrTAL2016, Dubrovnik, Croatia, 29 September - 1 October 2016.

Buchholz, Sabine and Marsi, Erwin (2006). CoNLL-X shared task on Multilingual Dependency Parsing. In Proceedings of the 10th Conference on Computational Natural Language Learning (CoNLL-X), pages 149–164, New York City, June 2006. © 2006 Association for Computational Linguistics

Fielding, Roy Thomas (2000). "Chapter 5: Representational State Transfer (REST)". *Architectural Styles and the Design of Network-based Software Architectures* (Ph.D.). University of California, Irvine.

Tufiș, Dan (2000). Using a Large Set of EAGLES-compliant Morpho-Syntactic Descriptors as a Tagset for Probabilistic Tagging. International Conference on Language Resources and Evaluation LREC'2000, Athens, 2000, pp. 1105-1112