Universitatea Alexandru Ioan Cuza din Iasi, Romania
Facultatea de Informatica

# Tehnologie de Automatizare pentru Asigurarea Mentenantei Software si pentru Administrarea Sistemelor

Rezumat Extins

*Autor:*
**Andrei Panu**

*Coordonator:*
**Prof. Dr. Henri Luchian**

Iunie 2017

**Tehnologie de Automatizare pentru Asigurarea Mentenantei Software si pentru Administrarea Sistemelor**


## Rezumat

Utilizarea in continua crestere a serviciilor cloud a schimbat modul cum aplicatiile software sunt dezvoltate, instalate si rulate. In zilele noastre aplicatiile sunt, in foarte multe cazuri, compuse din servicii multiple ce ruleaza in cloud, in diferite medii. In aceste medii sunt instalate de asemenea si aplicatii monolitice, de dimensiuni variate. Pana si aplicatiile mobile ce sunt instalate local utilizeaza servicii backend gazduite in cloud. Toate acestea ruleaza in anumite medii preconfigurate. Dupa dezvoltarea si instalarea initiala, mentinerea aplicatiilor si a mediilor acestora de executie actualizate vine cu unele provocari. Administratorii de sistem nu cunosc detalii interne despre aplicatiile care ruleaza pe infrastructura lor, astfel, in cazul existentei unei actualizari pentru interpretorul sau pentru o biblioteca de care depind una sau mai multe aplicatii, acestia nu vor sti daca versiunea noua va aduce unele modificari ce vor afecta buna functionare a aplicatiilor. Echipa de dezvoltare trebuie sa evalueze modificarile si sa ofere suport pentru versiunea noua. Asemenea sarcini necesita timp pentru realizarea lor. Problema este mai mare in cazul limbajelor interpretate, deoarece erorile apar in timpul executiei, numai atunci cand anumite secvente de cod sunt executate, si nu initial la compilare, cum este cazul limbajelor compilate. In aceasta teza ne axam pe cazul aplicatiilor dezvoltate folosind limbaje interpretate. Propunem o metoda noua care consta in analiza automata a aplicatiilor si verificarea automata daca schimbarile dintr-o versiune noua a unei dependente software le afecteaza. Scopul este de a imbunatati productivitatea in cazul administrarii aplicatiilor software si in vederea asigurarii mentenantei software. Pentru a realiza acest lucru, am proiectat un proces ce modeleaza abordarea umana de invatare din manuale a functionalitatilor oferite de diverse interpretoare sau biblioteci. Apoi, am proiectat o solutie ce automatizeaza acest proces, indiferent de limbajul de programare utilizat, folosind tehnici de invatare automata si de procesare a limbajului natural. In cele din urma, am dezvoltat si evaluat o platforma care implementeaza metoda noastra.



**Cuvinte cheie:** Extragerea Informatiilor, Identificarea Entitatilor cu Nume, Ingineria Cunostintelor, Ontologii, Invatare Automata, Web Mining, Web-ul Semantic

# Cuprins

# Lista Lucrari Stiintifice

1. Andrei Panu. A Novel Method for Improving Productivity in Software Administration and Maintenance. In *Proceedings of the 12th International Conference on Software Technologies* (ICSOFT 2017), Spain, July 24–26, 2017. (accepted for publication) (cat. B [1])

2. Lenuta Alboaie, Sinica Alboaie, Andrei Panu. Levels of Privacy for eHealth Systems in the Cloud Era. In *Proceedings of the 24th International Conference on Information Systems Development* (ISD 2015), pp. 243-252, Harbin, China, 2015. (cat. A [1])

3. Serban Ungureanu, Andrei Panu, Lenuta Alboaie. A Cloud Gaming System Design Insights. In *Proceedings of the 12th International Conference on e-Business* (ICE-B 2015), pp. 144-151, Colmar, France, 2015. (cat. B [1])

4. Sabin C. Buraga, Andrei Panu. A Web Tool for Extracting and Viewing the Semantic Markups. In *Proceedings of the 6th International Conference on Knowledge Science, Engineering and Management* (KSEM 2013), pp. 570-579, China, 2013. (cat. B [1])

5. Andrei Panu, Sabin C. Buraga, Lenuta Alboaie. Qsense – Learning Semantic Web Concepts by Querying DBpedia. In *Proceedings of the 10th International Conference on E-Business* (ICE-B 2013), pp. 351-356, Iceland, 2013. (cat. B [1])

6. Claudia Gheorghiu, Andrei Panu, Lenuta Alboaie. A Semantic Web Platform for Legal Knowledge in Cloud. In *Proceedings of the 12th International Conference on Intelligent Software Methodologies*, Tools and Techniques (SOMET 2013), pp. 121-126, Hungary, 2013. (cat. B [1])

7. Lenuta Alboaie, Sinica Alboaie, Andrei Panu. Swarm Communication – a Messaging Pattern Proposal for Dynamic Scalability in Cloud. In *Proceedings of the 15th IEEE International Conference on High Performance Computing and Communications* (IEEE HPCC 2013), pp. 1930-1937, China, 2013. (cat. B [1])

8. Alina Gozman (Munteanu), Daniel Munteanu, Andrei Panu, Lenuta Alboaie, Sabin-Buraga. SINUX - Ubuntu spiced up with Semantic Web, *10th Networking in Education and Research - RoEduNet IEEE Conference*, ISSN 2247-5443, pp.199-204, Romania, 2011. (cat. D [1])

[1] Conform clasificarilor CORE2013, CORE2014 si CORE2017.

# 1 [redacted]

# Prezentarea Tezei

## Contents



Utilizarea in continua crestere a serviciilor cloud a schimbat modul cum aplicatiile software sunt dezvoltate, instalate si rulate. In zilele noastre aplicatiile sunt, in foarte multe cazuri, compuse din servicii multiple ce ruleaza in cloud, in diferite medii. In aceste medii sunt instalate de asemenea si aplicatii monolitice, de dimensiuni variate. Pana si aplicatiile mobile ce sunt instalate local utilizeaza servicii backend gazduite in cloud. Toate acestea ruleaza in anumite medii preconfigurate. Dupa dezvoltarea si instalarea initiala, apar unele provocari pentru asigurarea mentenantei mediilor de executie si aplicatiilor. De exemplu, administratorii de sistem se confrunta cu o dilema atunci cand este disponibila o actualizare a interpretorului unui anumit limbaj (PHP, Python, etc.), mai ales daca este una majora. Ei nu sunt dezvoltatorii aplicatiilor sau serviciilor gazduite ce se bazeaza pe interpretor, astfel ca nu stiu daca actualizarea aduce modificari care vor afecta anumite functionalitati ale aplicatiilor. Nici nu este in responsabilitatea lor sa cunoasca detalii interne despre aplicatii. Daca dezvoltatorii au sarcina de a suporta o versiune noua a interpretorului, acestia trebuie sa faca o evaluare a modificarilor aduse de actualizare si a modificarilor care trebuie facute in aplicatie. Aceeasi problema apare in cazul existentei unei actualizari pentru o biblioteca de care aplicatia depinde. Aceste sarcini necesita efort si timp.

In aceasta teza propunem o metoda noua [20] ce ofera administratorilor informatii asupra a ce se va intampla daca se realizeaza actualizarea si dezvoltatorilor informatii despre ce modificari trebuie facute, toate acestea intr-un mod automat, imbunatatind astfel eficienta pentru aceste sarcini. Solutia noastra utilizeaza tehnici de invatare automata si de prelu-

crare a limbajului natural. Este independenta de limbajul de programare utilizat pentru dezvoltarea aplicatiilor.

## 1.1 Structura

Capitolul introductiv prezinta o scurta trecere in revista a situatiei actuale a mediilor cloud, a utilizarii inteligentei artificiale si a Internet of Things, evidentiind complexitatea mediilor si nevoia de automatizare. De asemenea, mentionam pe scurt problema adresata si introducem viziunea noastra. In acest capitol mai sunt prezentate obiectivele tezei si contributiile proprii.

Capitolul 2 prezinta conceptele si tehnologiile pe care se bazeaza solutia noastra. Este oferita o prezentare generala a tehnicilor de extragere a informatiilor si de identificare a entitatilor cu nume. Implementarea solutiei noastre ruleaza in mediile cloud si are la baza o arhitectura ce ii permite sa fie usor integrabila in medii diverse. Astfel, acest capitol mai contine prezentari generale asupra cloud computing si asupra tehnologiilor de integrare.

Capitolul 3 prezinta problema adresata si detaliaza abordarea noastra. De asemenea, contine o analiza a solutiilor similare.

Capitolul 4 prezinta platforma care implementeaza solutia noastra. Pe langa informatiile referitoare la proiectare si implementare, sunt prezentate si diverse rezultate experimentale si scenarii de instalare.

Capitolul 5 prezinta aspecte cheie referitoare la modelarea si partajarea de cunostinte si tehnologiile aferente, cu accent pe Web-ul Semantic. In acest capitol propunem o ontologie pentru domeniul adresat, utilizata in structurarea informatiilor extrase conform conceptelor definite.

Capitolul 6 prezinta diferite scenarii in mai multe arii in care solutia noastra are aplicabilitate. De asemenea sunt oferite informatii despre potentialele probleme de confidentialitate care pot aparea prin utilizarea solutiei.

In ultimul capitol prezentam concluziile si directiile viitoare de cercetare bazate pe ideile expuse in teza.

## 1.2 Contributii

In aceasta teza propunem o metoda noua menita sa imbunatateasca productivitatea in domeniul administrarii si al asigurarii mentenantei aplicatiilor software. Obiectivul principal este de a oferi un framework pentru optimizarea procesului de actualizare a aplicatiilor software. Modalitatea prin care se realizeaza acest lucru consta intr-o metoda originala care asista toate personale implicate in activitatile aferente, prin automatizarea unor sarcini specifice, reducand timpul necesar realizarii lor. Contributiile acestei teze pot fi rezumate dupa cum urmeaza:

- am proiectat un proces care modeleaza abordarea umana pentru invatarea din manuale a functionalitatilor oferite de interpretorul unui limbaj de programare sau de diverse biblioteci software;
- am proiectat o solutie care automatizeaza procesul si reduce timpul necesar rezolvarii problemei adresate, indiferent de limbajul de programare sau de bibliotecile utilizate;
- am dezvoltat si evaluat o platforma ce implementeaza solutia propusa;
- am propus o ontologie care modeleaza concepte din domeniul programarii, pe care o folosim pentru crearea bazei de cunostinte ce contine toate informatiile invatate de platforma;
- am propus o ierarhie cu entitati din domeniul programarii, definind tipuri noi de informatii ce pot fi identificate si extrase;
- oferim o perspectiva asupra modului in care solutia noastra poate fi aplicata si in alte arii: pentru Web-ul Semantic si pentru dezvoltarea software in contextul Internet of Things.

## 1.3 O Metoda Noua de Automatizare pentru Asigurarea Mentenantei Software si pentru Administrarea Sistemelor

In acest cadru vast de aplicatii software de dimensiuni diferite, care ruleaza in medii de executie configurate direct pe servere barebone, pe masini virtuale sau in containere software, managementul aplicatiilor si al mediilor de executie dupa dezvoltarea si instalarea initiala vine cu unele provocari. Propunerea noastra adreseaza o problema specifica de management, referitoare la actualizarea aplicatiilor.

Administratorii de sistem asigura buna functionare a infrastructurii pe care ruleaza aplicatii diferite. Una din sarcinile lor majore este sa mentina sistemele actualizate, ceea ce genereaza anumite dificultati. Atunci cand se confrunta cu situatia existentei unei actualizari pentru mediul de executie al aplicatiilor instalate, de exemplu pentru un interpretor, ei trebuie sa aiba un raspuns la intrebari precum:

- Vor mai rula aplicatiile existente pe noua versiune a interpretorului?
- Exista vreo parte a aplicatiilor care nu va mai rula din cauza modificarilor aduse de noua versiune?

Acestea nu sunt intrebari la care se poate raspunde cu usurinta, deoarece administratorul nu este, de obicei, dezvoltatorul aplicatiilor. Aceeasi situatie este valabila si in cazul bibliotecilor software instalate separat de aplicatii si de care acestea depind. Avand in

vedere ca administratorul nu are cunostinte interne despre aplicatii (cu exceptia versiunilor interpretorului si a bibliotecilor necesare atunci cand acestea au fost instalate initial), el se poate baza doar pe unele presupuneri in vederea realizarii actualizarii. O presupunere intuitiva ar fi daca exista o actualizare minora a versiunii, atunci totul ar trebui sa fie in regula, deoarece nu au fost facute schimbari majore in functionalitate. In majoritatea cazurilor, acest lucru este valabil, dar totusi este o presupunere, nu o certitudine. Problema apare atunci cand exista o actualizare majora, iar echipa de dezvoltare nu are in plan realizarea modificarilor incat versiunea noua sa fie suportata. Problema este foarte importanta mai ales in cazul limbajelor interpretate, deoarece erorile apar in momentul rularii, atunci cand anumite secvente de cod sunt executate, si nu initial, cand intreaga aplicatie este compilata, cum este cazul limbajelor compilate. Astfel, unele parti ale aplicatiei pot functiona, iar altele nu. Administratorul pur si simplu nu va sti daca vor exista parti ale aplicatiei care nu vor rula, el nu este dezvoltatorul si nici tester-ul. Aceasta problema scaleaza, caci un singur administrator poate avea mai multe aplicatii ce ruleaza pe infrastructura gestionata de acesta. Propunerea noastra ofera o solutie pentru aceasta situatie, analizand in mod automat aplicatiile si furnizand informatii referitoare la modificarile aduse de noua versiune si daca acestea vor afecta buna functionalitate a aplicatiilor.

In prezent exista o schimbare in ceea ce priveste managementul mediului de executie, de la administratorii de sistem la dezvoltatorii aplicatiilor (DevOps), prin utilizarea masinilor virtuale sau a containerelor software (de ex. Docker). Acest lucru nu rezolva problema, doar o paseaza dezvoltatorilor, desi asta nu inseamna ca administratorii nu mai sunt interesati de situatia masinilor virtuale sau a containerelor ce ruleaza pe infrastructura lor. Astfel, atunci cand echipa de dezvoltare trebuie sa suporte o versiune noua a interpretorului sau a unei biblioteci folosite, se confrunta cu aceeasi problema descrisa anterior. Chiar daca sunt cunoscute toate aspectele interne ale aplicatiei, nu se stie exact ce secvente de cod vor executa si care nu. Trebuie analizat changelog-ul sau ghidul de migrare si evaluat care sunt modificarile ce trebuie facute. Aceasta procedura manuala necesita mult timp. Propunerea noastra ajuta la reducerea timpului necesar.

Solutia pe care o propunem pentru rezolvarea acestei probleme consta intr-o tehnologie capabila sa scaneze automat codul sursa si sa verifice daca functionalitatile utilizate sunt suportate inca in versiunea noua a interpretorului/bibliotecii. Aceasta tehnologie este independenta de limbajul de programare utilizat. Dupa cum am mentionat anterior, ne axam pe limbajele interpretare. Tehnologie este compusa din trei module:

1. O aplicatie care analizeaza automat codul sursa, extrage functionalitatile folosite si interogheaza o baza de cunostinte ce ofera raspuns la intrebari precum: functionalitatea $X$ este suportata in noua versiune $N$? Daca nu, care sunt schimbarile care au fost facute?

2. O baza de cunostinte [18] creata automat ce contine informatii despre functionalitatile suportate in fiecare versiune a interpretorului/bibliotecii;

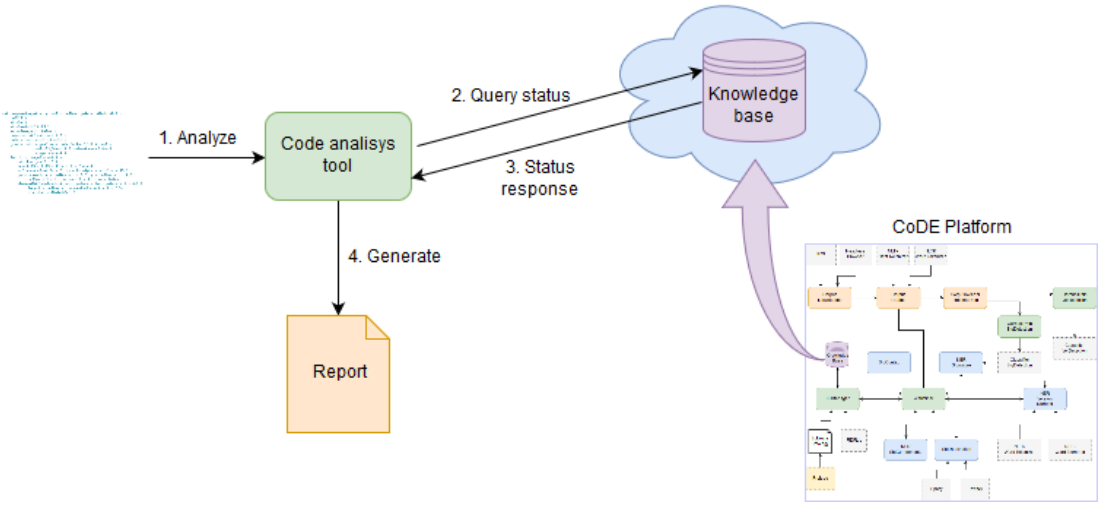


Figura 1-1: Workflow-ul General.

3. O platforma care extrage entitati specifice din manualele online sau offline, independent de limbajul de programare, si populeaza baza de cunostinte.

Procesul general este prezentat in Figura 1-1. Aplicatia de analiza este modulul care trebuie sa aiba acces la codul sursa si care utilizeaza baza de cunostinte pentru a verifica daca exista functionalitati ce nu mai sunt suportate in versiunea noua sau sunt marcate pentru a fi eliminate in viitor. Un raport este generat conform constatarilor. Aspectul principal al tehnologiei noastre il reprezinta informatiile continute in baza de cunostinte si modul cum sunt acestea obtinute. Pentru versiunea curenta a platformei care populeaza baza de cunostinte, functionalitatile luate in considerare sunt functiile suportate in toate versiunile interpretorului/bibliotecii (API-ul oferit). Astfel, aplicatia de analiza trebuie sa extraga toate functiile din cod, sa le elimine pe cele declarate local si sa interogheze baza de cunostinte pentru a verifica suportul in versiunea noua. Singura functionalitate care este mai complexa este filtrarea functiilor oferite de o anumita biblioteca sau de interpretor.

In continuare ne axam pe platforma care populeaza baza de cunostinte. Dupa cum am mentionat, informatiile continute constau in detalii privind toate functiile suportate de interpretor/biblioteca. Pentru fiecare functie, extragem atribute precum componentele signaturii (numele, numarul de argumente, tipul de date al argumentelor, ordinea argumentelor), tipul de date al valorii returnate, descrierea signaturii, disponibilitatea acesteia (daca mai este suportata, daca a fost eliminata, sau daca a fost marcata pentru a fi eliminata in viitor) si versiunea interpretorului/bibliotecii in care este suportata sau in care a fost eliminata. Toate informatiile sunt extrase din manualele online disponibile pe Web sau din cele offline. Platforma pe care am proiectat-o si implementat-o este capabila sa extraga in mod automat informatiile necesare, independent de continut (anumite manuale pentru anumite limbaje/biblioteci) si de structura paginilor Web. Suporta orice manual pentru orice limbaj. Singura restrictie se refera la sintaxa utilizata pentru scrierea functiilor. Aceasta

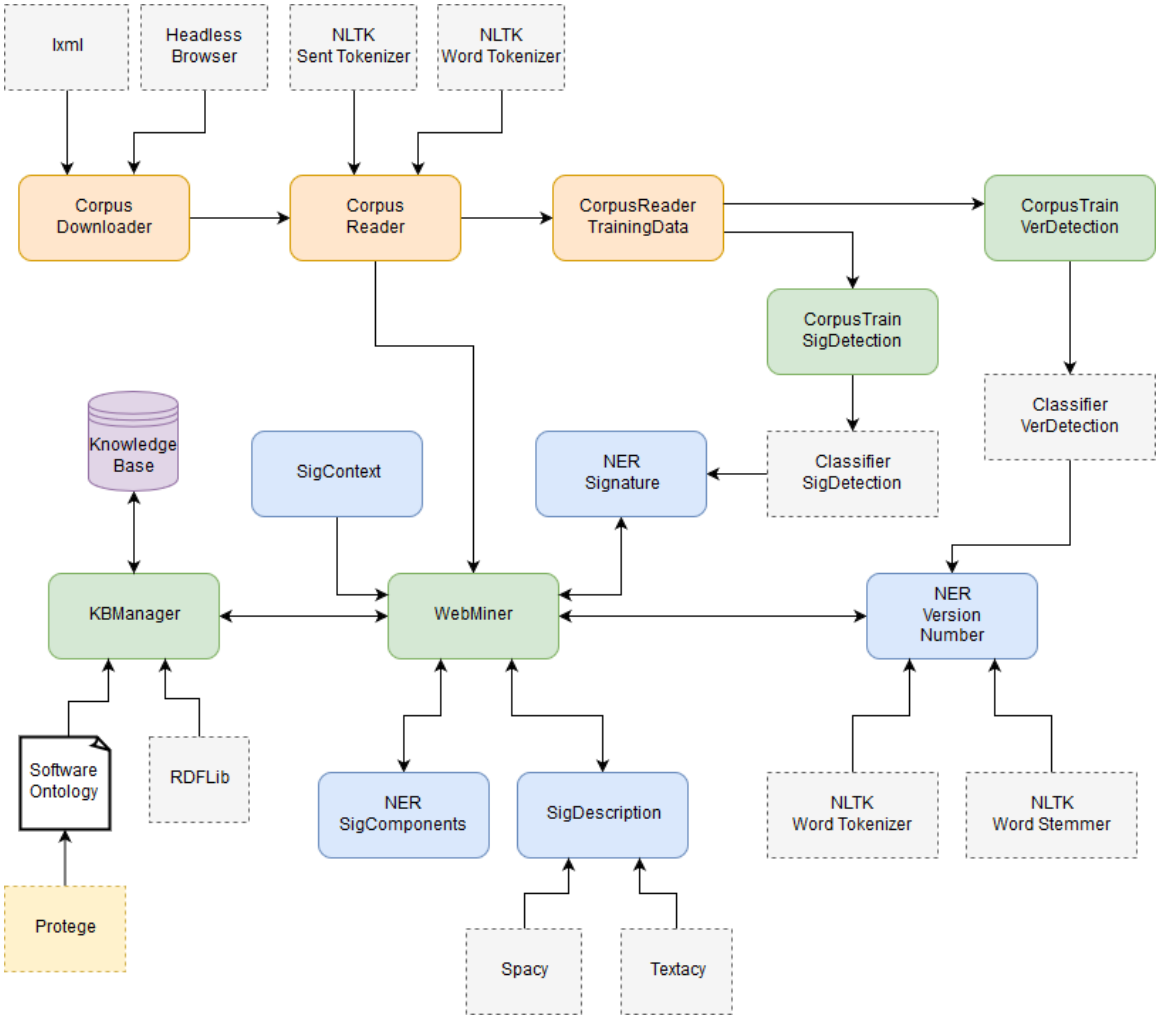


Figura 1-2: Arhitectura Platformei CoDE.

capabilitate este realizata folosind algoritmi de invatare automata si tehnici de procesare a limbajului natural [23, 9].

## 1.4 CoDE – Platforma de Extragere a Informatiilor

In aceasta sectiune prezentam arhitectura platformei care este capabila sa acceseze automat informatii de pe Web, sa identifice si sa extraga entitati specifice si sa populeze baza de cunostinte. Componentele sale sunt decuplate si independente de context. Fiecare componenta a fost proiectata sa faca parte dintr-o arhitectura distribuita. Platforma este implementata in Python. Figura 1-2 prezinta arhitectura sistemului. Platforma are patru componente principale, *CorpusTrain SigDetection*, *CorpusTrain VerDetection*, *WebMiner* si *KBManager*, iar functionalitatea sa este impartita in doua etape, una de antrenare si una de extragere a informatiilor.

In prima etapa antrenam doi clasificatori Naive Bayes, unul pentru detectarea signa-

turilor si unul pentru detectarea versiunilor din paginile de manual. Desi este unul din cele mai simple modele, am ales Naive Bayes pentru ca este cunoscut ca ofera rezultate bune in diverse cazuri de utilizare si pentru ca nu necesita o cantitate mare de date de antrenare. Am creat manual doua seturi de date pentru antrenare.

Pentru clasificatorul care detecteaza signaturi, setul de antrenare contine 279 de exemple pozitive si 296 negative, ce au fost extrase din manualele PHP si Python. Instantele pozitive sunt propozitii intregi ce contin signaturi. Instantele negative sunt propozitii ce mentioneaza cel putin o functie sau ce contin diferite secvente de cod existente in manuale. Au fost alese in mod special astfel pentru a reduce rata pozitivelor false. Primele 70% din instantele adnotate pentru fiecare eticheta sunt folosite pentru antrenare, iar restul pentru testare. Modelul antrenat are o acuratete de 98% pe setul de date de testare. Componentele *CorpusReader* si *CorpusReader TrainingData* sunt utilizate pentru incarcarea datelor de antrenare in memorie. *CorpusTrain SigDetection* creeaza seturile de proprietati, imparte datele si antreneaza clasificatorul, generand *Classifier SigDetection*. Modelul antrenat este folosit mai departe de componenta *NER Signature*, care ofera un singur serviciu: primeste o lista de propozitii si le identifica pe cele care reprezinta sigaturi.

Pentru clasificatorul care detecteaza versiunile, instantele de antrenare sunt create din exemple preluate din manualele PHP, Python si jQuery. Numarul total de instante este de 518, din care 62 pozitive si 456 negative. Setul negativ contine numere diferite care nu reprezinta o versiune. Seturile de proprietati sunt impartite, primele 70% din instantele adnotate pentru fiecare eticheta sunt folosite pentru antrenare, iar restul pentru testare. Modelul are o acuratete de 95% pe datele de testare. Aceleasi componente mentionate anterior sunt utilizate pentru incarcarea datelor de antrenament in memorie. *CorpusTrain VerDetection* este componenta care creeaza seturile de proprietati, face divizarea si antreneaza clasificatorul, generand *Classifier VerDetection*. Modelul antrenat este utilizat de componenta *NER VersionNumber*, care ofera un singur serviciu: primeste signaturile impreuna cu contextele descriptive ale acestora si identifica cuvintele care reprezinta versiuni.

Cea de-a doua etapa consta in accesarea informatiilor disponibile online, analizarea lor si extragerea cunostintelor specifice. Orchestratorul principal al tuturor operatiilor este componenta *WebMiner*. Celelalte componente implicate sunt *CorpusDownloader*, *CorpusReader*, *NER Signature*, *SigContext*, *NER VersionNumber*, *NER SigComponents*, *SigDescription*. Datele extrase sunt apoi salvate in baza de cunostinte, operatie realizata de *KBManger*. *CorpusDownloader* este componenta care lucreaza cu paginile online de manual ce contin datele dorite. Continutul este accesat prin intermediul unui browser *headless*, furnizand informatiile exact asa cum sunt randate intr-un browser si vazute de o persoana, fara niciun cod (de ex. elemente HTML, proprietati CSS, cod JavaScript). In continuare, datele sunt preprocesate, fiind identificate propozitiile, iar in cadrul fiecarei propozitii, cuvintele. Aceste operatii sunt realizate de *CorpusReader*. *WebMiner* trimite mai departe informatiile componentei *NER Signature*, care identifica signaturile existente utilizand clasificatorul antrenat. Fiecare signatura este apoi trimisa componentei *NER*

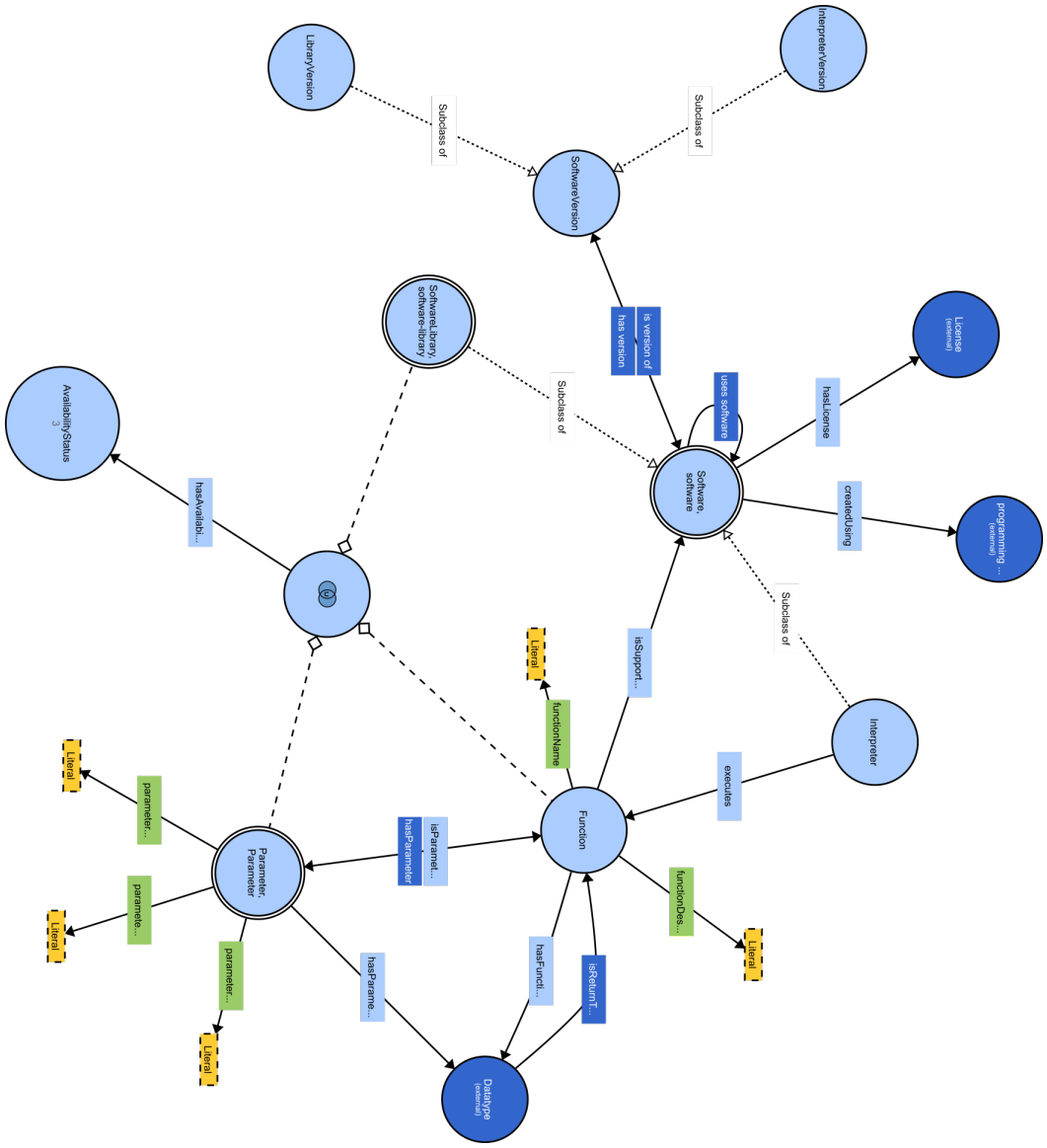


Figura 1-3: Reprezentarea vizuala a ontologiei ProgO (imagine generata de VOWL [14])

*SigComponents* pentru a extrage urmatoarele elemente: numele signaturii, tipul de date returnat, parametrii. Pentru fiecare parametru, extragem numele, tipul de date si pozitia. Acest lucru se realizeaza utilizand reguli create manual, bazate pe expresii regulate. Toate aceste date sunt trimise ulterior componentei *NER SigContext*, care identifica contextele descriptive ale fiecarei signaturi, prin utilizarea unor reguli create manual. Definim un context descriptiv ca lista propozitiilor ce contin detalii referitoare la o singura signatura. Contextul descriptiv al fiecarei signaturi este trimis componentei *NER VersionNumber* pentru a extrage versiunile in care functia este suportata, eliminata sau marcata pentru eliminare in viitor. Aceasta componenta utilizeaza clasificatorul antrenat si cateva reguli create manual care verifica contextul cuvantului ce reprezinta o versiune pentru iden-

tificarea statusului (disponibila, eliminata, marcata pentur eliminare). Ultimul pas consta in extragerea descrierii functiei. Este realizat de componenta *SigDescription* ce analizeaza sintactic propozitiile si identifica descrierea pe baza unor reguli create manual ce modeleaza diferite sabloane observate, care sunt folosite in formularea descrierilor. Toate informatiile extrase sunt furnizate componentei *KBManager*, care gestioneaza baza de cunostinte.

Componenta *KBManager* primeste informatiile extrase si genereaza instante pentru baza de cunostinte. Datele sunt exprimate folosind modelul RDF (Resource Description Framework) [1]. Triplele RDF contin informatii structurate conform conceptelor ilustrate in ontologia ProgO, ontologie [18, 11] pe care am proiectat-o special pentru domeniul programarii. Ontologiile existente specifice acestui domeniu [15, 31, 19] nu contin toate descrierile conceptuale necesare cazului nostru. Ontologia ProgO poate fi clasificata ca o ontologie la nivel aplicatie [12], insemnand ca a fost conceputa special pentru a fi utilizata ca parte a solutiei noastre. In procesul de proiectare a unei astfel de ontologii, a trebuit sa tinem cont de doua cerinte: reutilizarea a cat se poate de multe concepte si proprietati din ontologiile existente si luarea in considerare a faptului ca orice clase sau proprietati noi adaugate trebuie sa se potriveasca domeniului aplicatiei.

Domeniul ontologiei ProgO vizeaza reprezentarea partilor unui cod sursa cu intentia de a gestiona functiile expuse si aspecte precum disponibilitatea acestora, parametrii si tipurile de date returnate. Astfel, cateva dintre conceptele importante modelate de ontologia noastra constau in: numele functiei, tipul de date returnat, descrierea functiei, numele parametrului, tipul de date al parametrului, pozitia parametrului, descrierea parametrului si numarul versiunii si disponibilitatea functiei. Reprezentarea ontologiei este prezentata in Figura 1-3.

Ontologia a fost creata utilizand Protege [2]. Datele sunt stocate intr-un Triple Store furnizat de Virtuoso [3].

## 1.5 Rezultate Experimentale

Am realizat teste pe diverse manuale online pentru a evalua performanta generala a platformei in extragerea tuturor informatiilor necesare. Am directionat platforma catre pagini selectate aleatoriu din manualele Node.js (version 7.7.0) [4], Ruby [5], PHP [6], Python (version 3.6.0) [7] si Laravel (version 5.4) [8].

Tabelul 1.1 sumarizeaza rezultatele obtinute de platforma in detectarea signaturilor pentru fiecare caz. A doua coloana contine numarul total de signaturi existente in pagina de manual, iar ultima coloana procentul signaturilor detectate. Am obtinut urmatoarele

[1] https://www.w3.org/RDF/
[2] http://protege.stanford.edu/
[3] https://virtuoso.openlinksw.com/
[4] https://nodejs.org/api/util.html
[5] https://ruby-doc.org/docs/ruby-doc-bundle/Manual/man-1.4/function.html
[6] http://php.net/manual/en/function.chmod.php
[7] https://docs.python.org/3/library/os.path.html
[8] https://laravel.com/docs/5.4/helpers

Tabelul 1.1: Performanta in detectarea signaturilor.

| **Manual** | **Nr. de functii** | **Rata detectiei** |
|---|---|---|
| Node.js | 26 | 100% |
| Ruby | 59 | 64.4% |
| PHP | 1 | 100% |
| Python | 30 | 100% |
| Laravel | 80 | 98.75% |

rezultate:

- In cazul Node.js a extras toate signaturile, fara pozitive false, fiind capabila sa filtreze toate celelalte functii existente in exemplele de cod din pagina;
- In cazul PHP a extras cu succes singura signatura existenta. De asemenea a mai identificat 5 functii (pozitive false), deoarece pagina contine foarte multe comentarii cu exemple de cod, platforma nereusind sa filtreze toate functiile mentionate in zonele respective;
- In cazul Python a extras toate signaturile, fara pozitive false, fiind capabila sa filtreze toate functiile mentionate in exemplele de cod;
- In cazul Laravel nu a reusit sa identifice o singura signatura, a carei nume contine un singur caracter (functia *e()*). Nu avem pozitive false;
- In cazul Ruby a detectat doar 38 de functii din totalul de 59. Acest lucru se datoreaza faptului ca in pagina respectiva exista multe signaturi care nu sunt scrise utilizand paranteze (de ex. *at_exit*, *binding*, *chop*, *fork*), aceasta fiind o caracteristica foarte importanta pentru clasificatorul antrenat.

Pentru fiecare dintre signaturile identificate, componenta *NER SigComponents* a extras cu succes toate elementele lor (numele, tipul de date returnat, parametrii, etc.). In ceea ce priveste extragerea versiunilor, am obtinut urmatoarele rezultate:

- pentru Node.js: pagina contine 26 de signaturi, 5 dintre ele avand specificata o singura versiune reprezentand cand a fost adaugata, restul continand doua versiuni, reprezentand cand a fost adaugata si cand a fost marcata pentru eliminare. Platforma a detectat corect toate versiunile, cu statusul aferent. De asemenea a identificat si versiunea generala (*7.10.0*) mentionata in afara contextului descriptiv al signaturilor, avand o prioritate mai scazuta fata de celelalte. Nu exista cazuri de pozitive sau negative false;
- pentru Ruby: pagina nu contine nicio versiune, astfel platforma in mod corect nu a identificat nimic;

Tabelul 1.2: Performanta in detectarea descrierilor.

| **Manual** | **Nr. signaturilor detectate** | **Rata detectiei** |
|---|---|---|
| Node.js | 26 | 76.9% |
| Ruby | 38 | 68.4% |
| PHP | 6 | 100% |
| Python | 30 | 93.3% |
| Laravel | 79 | 96.2% |

- pentru PHP: pagina contine o singura signatura, avand mentionate 3 versiuni (*PHP 4*, *PHP 5*, *PHP 7*). Platforma le-a identificat cu succes pe toate;
- pentru Python: pagina contine 30 de signaturi, 4 dintre ele avand specificate doua versiuni, 1 dintre ele avand specificate 3 versiuni, iar restul o singura versiune. Platforma a identificat in mod corect toate versiunile, fara pozitive sau negative false. A identificat de asemenea si versiunea generala (*3.6.0*) existenta in afara contextului descriptiv al signaturilor;
- pentru Laravel: pagina nu contine nicio versiune in contextele descriptive ale functiilor, astfel ca platforma, in mod corect, nu a identificat niciuna. A identificat versiunea generala (*5.4*) mentionata in afara contextelor.

In cazul detectarii descrierilor signaturilor, platforma a obtinut rezultatele prezentate in Tabelul 1.2. A doua coloana reprezinta numarul de signaturi detectate (fiecare avand o singura descriere), iar ultima coloana procentul signaturilor detectate. In cazul Node.js nu a reusit sa identifice 6 descrieri. In cazul Ruby, nu a reusit sa identifice 12 descrieri. In cazul PHP, a identificat cu succes descrierea signaturii. Pentru celelalte 5 functii (pozitive false) nu a detectat nicio descriere, pentru ca acestea sunt exemple de cod si nu au asa ceva. In cazul Python, nu a reusit sa identifice descrierile a 2 signaturi. In cele din urma, in cazul Laravel nu a reusit sa identifice 3 descrieri. Aceste omisiuni sunt din cauza faptului ca modul de exprimare al descrierilor nu corespunde cu modelele cunoscute de catre algoritmul de identificare sau din cauza ca algoritmul de analiza sintactica nu detecteaza corespunzator functiile gramaticale ale cuvintelor.

## 1.6 Solutii Asemanatoare

### Verificarea Compatibilitatii Codului

Metoda standard de actualizare a mediului de executie al unei aplicatii (de ex. interpretorul sau unele biblioteci de care depinde) consta in configurarea unui mediu de testare cu noile versiuni si verificarea intregii aplicatii. Aceasta este, teoretic, cea mai buna metoda, deoarece este realizata de dezvoltatorii care cunosc foarte bine aplicatia si testeaza toate functionalitatile acesteia. Scopul este sa se asigure ca toate secventele de cod ajung sa fie

executate. Dezavantajele constau in faptul ca este consumatoare de timp si este predispusa la erori umane (de ex. se poate omite verificarea unor functionalitati). Timpul necesar acestei activitati poate fi redus sau eliminat complet prin existenta testelor automate. Din pacate, situatiile in care teste automate sunt dezvoltate pentru intreaga aplicatie sunt rare. De obicei doar functionalitatile principale ale aplicatiei sunt acoperite de teste automate, celelalte ramanand pentru testarea manuala.

Exista unele instrumente care au fost dezvoltate pentru automatizarea verificarii compatibilitatii, dar ele sunt limitate ca acoperire si sunt dependente de limbaj. De exemplu, pentru PHP, exista *PHP CodeSniffer* [9] care tokenizeaza codul sursa si detecteaza si repara incalcarile de formatare a codului, fara a-l executa. Aceasta aplicatie este utilizata mai departe de *PHPCompatibility* [10], ce contine un set de reguli pentru verificarea compatibilitatii intre versiuni diferite ale PHP. Acest instrument genereaza un raport ce contine toate problemele identificate. Comparativ cu solutia noastra, avantajul lui este ca poate identifica mai multe tipuri de schimbari aparute, nu doar la nivel de API, dar nu absolut totul. Dezavantajele constau in faptul ca baza de cunostinte este creata manual, suporta doar functionalitatile oferite de interpretorul PHP si doar schimbarile facute incepand cu versiunea 5.0 a acestuia. Alte instrumente similare au fost dezvoltate pentru diverse limbaje si biblioteci, precum *jquery-migrate* [11] pentru migrarea la versiunea 3.0+, *wp-deprecated-checker* [12], ce verifica functiile oferite de Wordpress, *2to3* [13], un program de conversie automata a codului Python din versiunea 2.0 in versiunea 3.0, *PHP7MAR* [14] sau *php7cc* [15], ce sunt instrumente de migrare la versiunea PHP 7, *depcheck* [16], o alta aplicatie de verificare a utilizarii functiilor eliminate. Dezavantajele acestor instrumente sunt ca au acoperire restransa si sunt dezvoltate doar pentru un anumit limbaj sau biblioteca. Mai multe astfel de solutii pot fi gasite pentru diverse limbaje/biblioteci, dar nu chiar pentru toate.

Mai exista de asemenea IDE-uri care ofera instrumente de analiza a codului. Un astfel de exemplu este *PhpStorm* [17]. Contine un analizor static de cod ce poate detecta diferite ineficiente in cod. Cele mai des acoperite situatii sunt detectarea unor potentiale bug-uri, identificarea unor secvente de cod care nu vor fi executate niciodata, detectarea unor potentiale probleme de performanta, incalcarea unor reguli si standarde privind scrierea codului, verificarea conformitatii cu specificatiile. Toate aceste verificari sunt create manual. Suportul pentru detectarea problemelor de compatibilitate a codului in cazul diferitelor versiuni ale interpretorului PHP este limitat, constrans de existenta unor astfel de verificari

[9] http://pear.php.net/package/PHP_CodeSniffer/
[10] https://github.com/wimg/PHPCompatibility
[11] https://github.com/jquery/jquery-migrate
[12] https://gist.github.com/jbuchbinder/7419000
[13] https://docs.python.org/3/library/2to3.html
[14] https://github.com/Alexia/php7mar
[15] https://github.com/sstalle/php7cc
[16] https://www.phpclasses.org/package/9084-PHP-Find-deprecated-functions-and-suggest-replacements.html
[17] https://www.jetbrains.com/phpstorm/

Tabelul 1.3: Compararea solutiilor asemanatoare pentru verificarea compatibilitatii.

| **Solutie/Criteriu** | **independenta de limbaj** | **suport pentru biblioteci** | **crearea automata a BD** | **verificarea functiilor** | **verificarea altor modificari** |
|---|---|---|---|---|---|
| Solutia noastra | ✓ | ✓ | ✓ | ✓ | × |
| PHPCompatibility | × | × | × | ✓ | ✓ |
| jquery-migrate | × | × | × | ✓ | ✓ |
| wp-deprecated-checker | × | × | × | ✓ | ✓ |
| 2to3 | × | × | × | ✓ | ✓ |
| PHP7MAR | × | × | × | ✓ | ✓ |
| php7cc | × | × | × | ✓ | ✓ |
| depcheck | × | × | × | ✓ | × |
| PhpStorm | × | × | × | × | ✓ |
| PHPMD | × | × | × | × | ✓ |
| PyLint | × | × | × | × | ✓ |

create manual. Nu se axeaza pe oferirea unei astfel de functionalitati. O alta categorie de instrumente care analizeaza codul sunt *linter*-ele (de ex. PHPMD [18], Pylint [19]). Si in cazul acestora, accentul se pune pe verificarea diferitelor incalcari ale standardelor de scriere a codului si a diferitelor erori in modul in care este scris codul, nu a verificarii utilizarii de functii care au fost eliminate sau marcate pentru a fi eliminate. Tabelul 1.3 sumarizeaza concluziile noastre, conform urmatoarelor criterii:

- independenta de limbaj – daca solutia suporta limbaje multiple sau a fost dezvoltata pentru unul singur;
- suport pentru biblioteci – daca solutia verifica functionalitatile oferite de mai multe biblioteci si nu doar de una singura;
- crearea automata a bazei de cunostinte – daca sursa de informatii a solutiei este creata manual sau automat;
- verificarea functiilor – daca solutia este capabila sa verifice daca functiile utilizate mai sunt suportate;
- verificarea altor modificari – daca solutia poate verifica alte schimbari care apar (de sintaxa, termeni, etc.).

Multe cercetari sunt realizate in scopul automatizarii verificarii compatibilitatii componentelor software cu versiuni mai vechi, dar solutiile nu ofera aceeasi functionalitate ca propunerea noastra, ci una complementara [21, 29, 13, 27]. Sursele lor de informatii sunt "repository"-urile software, pe care le monitorizeaza si analizeaza. Abordarile se bazeaza pe tehnici diferite care evalueaza diferentele dintre codul sursa vechi si cel nou (intregul

[18] https://phpmd.org/
[19] https://www.pylint.org/

cod sau numai interfetele). Accectul se pune pe evaluarea compatibilitatii unei versiuni noi a unei componente software cu versiunea veche, din punct de vedere functional. Nu se mentioneaza nimic referitor la capacitatea de a furniza informatii despre modificari privind suportarea sau eliminarea unor functii in diferitele versiuni ale interpretoarelor/bibliotecilor. De asemenea, modul lor de functionare este prin compararea a doua "repository"-uri software, ceea ce nu este cazul nostru.

### Extragerea Entitatilor din Domeniul Programarii

Propunerea noastra are si o contributie cu privire la posibilitatea identificarii si extragerii entitatilor cu nume in domeniul programarii. In [16] este realizata o prezentare generala asupra diferitelor tipuri de entitati suportate pentru activitatea NER (Named Entity Recognition). Termenul "named entity" a fost utilizat prima data in cadrul MUC-6 (Sixth Message Understanding Conference) [10]. Activitatile de extragere s-au axat pe identificarea numelor proprii (de ex. nume de persoane, nume de organizatii, nume de locatii) si expresiilor numerice (de ex. ora, data, bani, procente). In ceea ce priveste recunoasterea numelor [3], cele trei exemple mentionate reprezinta cele mai studiate tipuri de entitati, clasificate ca "enamex". Mai departe au fost abordate diferite subcategorii, precum "politician" si "entertainer" pentru categoria "persoane" [8], sau "oras", "stat", "tara" pentru "locatie" [7]. Conferintele CONLL utilizeaza tipul "diverse" pentru a include numele proprii care nu se incadreaza in categoria "enamex". Alte categorii create in cadrul MUC sunt "timex", ce include tipuri de entitati precum "data" si "timp", si "numex", ce include tipuri precum "bani", "procente". Pentru nevoi specifice, au fost definite tipuri marginale, precum "film" si "om de stiinta" [5], "email" si "numar de telefon" [30], "domeniu de cercetare" si "nume proiect" [32], "titlu carte" [2, 30], "denumire job" [4]. Multe cercetari in aceasta directie se realizeaza si pentru domeniul bioinformaticii, fiind recunoscute tipuri precum "proteina", "AND", "ARN", "tip celula" [28, 26, 25]. Studii inrudite au fost realizate pentru detectarea numelor de droguri [22] si de substante chimice [17].

O alta categorie de sisteme NER, numita "open domain NERs" [1, 6], le include pe cele care nu au fost dezvoltate pentru anumite tipuri de entitati, ele nu restrictioneaza tipurile posibile pe care le pot extrage. Desi aceste sisteme pot fi utilizate teoretic pentru orice domeniu, toate experimentele si ajustarile au fost facute pentru cele mai cunoscute tipuri de entitati din domeniul general al stirilor. In plus, pentru contextul acestor solutii, a fost definita o ierarhie extinsa ce contine aproximativ 200 de categorii [24]. Au fost adaugate categorii precum culoare, animal, religie, substanta si subcategorii precum aeroport, muzeu, rau. Aceasta ierarhie este conceputa tot pentru domeniul general. Nu contine nimic referitor la domeniul programarii. Nu am reusit sa gasim nicio solutie care a fost dezvoltata si functioneaza cu entitati din domeniul adresat de noi. Astfel, in aceasta teza propunem o ierarhie de entitati ce contine categorii noi care acopera conceptele cu care solutia noastra lucreaza. Nu intentioneaza sa fie o ierarhie cuprinzatoare a intregului domeniu. In functie

Tabelul 1.4: Compararea solutiilor asemanatoare pentru extragerea entitatilor cu nume

| Solutie/Domeniu | Programare | General |
|---|---|---|
| CoDE | ✓ | × |
| IBM Watson (AlchemyAPI) | × | ✓ |
| OpenCalais | × | ✓ |
| MeaningCloud | × | ✓ |

de necesitati, va fi extinsa.

Solutiile comerciale cunoscute care ofera analiza pe text se axeaza de asemenea pe extragerea termenilor generali. Am testat mai multe aplicatii pe unele pagini alese aleatoriu din manualele PHP [20], Python [21] si Node.js [22]. Table 1.4 sumarizeaza rezultatele obtinute.

Modulul Natural Language Understanding al IBM Watson [23], ce a integrat AlchemyAPI, una din cele mai cunoscute solutii pentru procesarea limbajului natural, contine o lista larga de tipuri si subtipuri de entitati suportate [24]. Printre acestea, regasim cateva subtipuri aplicabile domeniului nostru, precum "ProgrammingLanguage", "ProgrammingLanguageDesigner", "Software", "SoftwareDeveloper", "SoftwareLicense". In cazurile noastre de testare, nu a fost identificat nimic de interes. Din pagina Node.js, au fost identificati termenii "len" ca fiind o persoana si "1 2 3" ca fiind o cantitate. Din pagina PHP nu a fost identificat nimic. Din pagina Python, a detectat termenii "unc" ca fiind o organizatie si "r" ca fiind o persoana. Avand in vedere tipurile suportate, consideram ca ar fi trebuit sa identifice cel putin termenii "PHP", "Node.js" si "Python".

O alta solutie foarte cunoscuta si utilizata este OpenCalais [25]. A detectat in mod corect limbajele de programare si alte termeni din domeniul IT, dar nimic relevant pentru scopul nostru.

MeaningCloud [26] ofera de asemenea servicii de analiza a textului. Demo-ul online limiteaza datele de intrare la 500 de caractere, prin urmare am selectat doar 8 functii din pagina de manual a Node.js si 8 functii din cea a Python, plus contextele descriptive ale acestora. Pagina PHP contine o singura functie, dar are foarte multe exemple de cod. Din aceasta pagina am selectat doar functia care ne intereseaza si contextul descriptiv al acesteia. Din pacate, la fel ca in cazul celorlalte solutii, nu a detectat nimic relevant scopului nostru.

Toate aceste rezultate sunt de asteptat, deoarece solutiile sunt concepute pentru domeniul general al stirilor, nu pentru domeniul specific adresat de noi. Ele sunt capabile sa recunoasca entitati din categorii precum persoane, locatii, date, companii, produse, nu entitati in contextul programarii.

[20] http://php.net/manual/en/function.chmod.php
[21] https://docs.python.org/3/library/os.path.html
[22] https://nodejs.org/api/util.html
[23] https://www.ibm.com/watson/developercloud/natural-language-understanding.html
[24] https://www.ibm.com/watson/developercloud/doc/natural-language-understanding/entity-types.html
[25] http://www.opencalais.com/
[26] https://www.meaningcloud.com/

# Bibliografie


[1] Enrique Alfonseca and Suresh Manandhar. An Unsupervised Method for General Named Entity Recognition And Automated Concept Discovery. In *In: Proceedings of the 1 st International Conference on General WordNet*, 2002. (Cited on page 14.)

[2] Sergey Brin. Extracting Patterns and Relations from the World Wide Web. In *The World Wide Web and Databases, International Workshop WebDB'98, Valencia, Spain, March 27-28, 1998, Selected Papers*, pages 172–183, 1998. (Cited on page 14.)

[3] Sam Coates-Stephens. The Analysis and Acquisition of Proper Names for the Understanding of Free Text. *Computers and the Humanities*, 26(5-6):441–456, 1992. (Cited on page 14.)

[4] William W. Cohen and Sunita Sarawagi. Exploiting dictionaries in named entity extraction: combining semi-Markov extraction processes and data integration methods. In *Proceedings of the Tenth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Seattle, Washington, USA, August 22-25, 2004*, pages 89–98, 2004. (Cited on page 14.)

[5] Oren Etzioni, Michael J. Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Unsupervised named-entity extraction from the Web: An experimental study. *Artif. Intell.*, 165(1):91–134, 2005. (Cited on page 14.)

[6] Richard Evans. A framework for named entity recognition in the open domain. *Recent Advances in Natural Language Processing III: Selected Papers from RANLP*, 260(267-274):110, 2003. (Cited on page 14.)

[7] Michael Fleischman. Automated Subcategorization of Named Entities. In *Association for Computational Linguistic, 39th Annual Meeting and 10th Conference of the European Chapter, Companion Volume to the Proceedings of the Conference: Proceedings of the Student Research Workshop and Tutorial Abstracts, July 9-11, 2001, Toulouse, France.*, pages 25–30, 2001. (Cited on page 14.)

[8] Michael Fleischman and Eduard H. Hovy. Fine Grained Classification of Named Entities. In *19th International Conference on Computational Linguistics, COLING 2002, Howard International House and Academia Sinica, Taipei, Taiwan, August 24 - September 1, 2002*, 2002. (Cited on page 14.)

[9] Ralph Grisham. Information Extraction: Capabilities and Challenges, 2012. (Cited on page 6.)

[10] Ralph Grishman and Beth Sundheim. Message Understanding Conference- 6: A Brief History. In *16th International Conference on Computational Linguistics, Proceedings*

*of the Conference, COLING 1996, Center for Sprogteknologi, Copenhagen, Denmark, August 5-9, 1996*, pages 466–471, 1996. (Cited on page 14.)

[11] Thomas R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowl. Acquis.*, 5(2):199–220, June 1993. (Cited on page 9.)

[12] N. Guarino. *Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy.* IOS Press, Amsterdam, The Netherlands, The Netherlands, 1st edition, 1998. (Cited on page 9.)

[13] Ahmed Hatim, Ali A.S.A. Elgamal, Hisham E. Elshishiny, and Mahmoud Rashad Ibrahim. Verification of backward compatibility of software components. https://www.google.com/patents/US20150169320?utm_source=gb-gplus-sharePatent, 2015. (Cited on page 13.)

[14] Steffen Lohmann, Stefan Negru, Florian Haag, and Thomas Ertl. Visualizing Ontologies with VOWL. *Semantic Web*, 7(4):399–419, 2016. (Cited on page 8.)

[15] James Malone, Andy Brown, Allyson Lister, Jon Ison, Duncan Hull, Helen Parkinson, and Robert Stevens. The software ontology (swo): a resource for reproducibility in biomedical data analysis, curation and digital preservation. In *Journal of Biomedical Semantics*, 2014. (Cited on page 9.)

[16] David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007. (Cited on page 14.)

[17] Meenakshi Narayanaswamy, K. E. Ravikumar, and K. Vijay-Shanker. A Biological Named Entity Recognizer. In *Proceedings of the 8th Pacific Symposium on Biocomputing, PSB 2003, Lihue, Hawaii, USA, January 3-7, 2003*, 2003. (Cited on page 14.)

[18] Natalya F. Noy, Deborah L. McGuinness, et al. Ontology development 101: A guide to creating your first ontology, 2001. (Cited on pages 4 and 9.)

[19] Daniel Oberle, Stephan Grimm, and Steffen Staab. An ontology for software. In *Handbook on ontologies*, pages 383–402. Springer, 2009. (Cited on page 9.)

[20] Andrei Panu. A novel method for improving productivity in software administration and maintenance. In *12th International Conference on Software Technologies (ICSOFT 2017), Madrid, Spain, July 24–26 (accepted for publication)*, 2017. (Cited on page 1.)

[21] A Ponomarenko and V Rubanov. Backward compatibility of software interfaces: Steps towards automatic verification. *Programming and Computer Software*, 38(5):257–267, 2012. (Cited on page 13.)

[22] Thomas C Rindflesch, Lorraine Tanabe, John N Weinstein, and Lawrence Hunter. EDGAR: extraction of drugs, genes and relations from the biomedical literature. In *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, page 517. NIH Public Access, 2000. (Cited on page 14.)

[23] Sunita Sarawagi. Information extraction. *Foundations and trends in databases*, 1(3):261–377, 2008. (Cited on page 6.)

[24] Satoshi Sekine and Chikashi Nobata. Definition, Dictionaries and Tagger for Extended Named Entity Hierarchy. In *Proceedings of the Fourth International Conference on Language Resources and Evaluation, LREC 2004, May 26-28, 2004, Lisbon, Portugal*, 2004. (Cited on page 14.)

[25] Burr Settles. Biomedical named entity recognition using conditional random fields and rich feature sets. In *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications*, pages 104–107. Association for Computational Linguistics, 2004. (Cited on page 14.)

[26] Dan Shen, Jie Zhang, Guodong Zhou, Jian Su, and Chew-Lim Tan. Effective adaptation of a hidden markov model-based named entity recognizer for biomedical domain. In *Proceedings of the ACL 2003 workshop on Natural language processing in biomedicine-Volume 13*, pages 49–56. Association for Computational Linguistics, 2003. (Cited on page 14.)

[27] Efstratios Tsantilis. Method and system to monitor software interface updates and assess backward compatibility. https://www.google.com/patents/US7600219, 2009. (Cited on page 13.)

[28] Yoshimasa Tsuruoka and Jun'ichi Tsujii. Boosting precision and recall of dictionary-based protein name recognition. In *Proceedings of the ACL 2003 workshop on Natural language processing in biomedicine-Volume 13*, pages 41–48. Association for Computational Linguistics, 2003. (Cited on page 14.)

[29] Yannick Welsch and Arnd Poetzsch-Heffter. Verifying backwards compatibility of object-oriented libraries using boogie. In *Proceedings of the 14th Workshop on Formal Techniques for Java-like Programs*, pages 35–41. ACM, 2012. (Cited on page 13.)

[30] Ian H Witten, Zane Bray, Malika Mahoui, and William J Teahan. Using language models for generic entity extraction. In *Proceedings of the ICML Workshop on Text Mining*, 1999. (Cited on page 14.)

[31] Michael Würsch, Giacomo Ghezzi, Matthias Hert, Gerald Reif, and Harald C. Gall. SEON: a pyramid of ontologies for software evolution and its applications. *Computing*, 94(11):857–885, 2012. (Cited on page 9.)

[32] Jianhan Zhu, Victoria S. Uren, and Enrico Motta. ESpotter: Adaptive Named Entity Recognition for Web Browsing. In *WM 2005: Professional Knowledge Management - Experiences and Visions, Contributions to the 3rd Conference Professional Knowledge Management - Experiences and Visions, April 10-13, 2005, Kaiserslautern, Germany*, pages 505–510, 2005. (Cited on page 14.)