UNIVERSITATEA POLITEHNICA BUCUREȘTI
FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE
CATEDRA DE AUTOMATICĂ ȘI INFORMATICĂ APLICATĂ

Ing. Irina RANCEA

Sistem automat de generare de cod pentru parsere bazat pe procesare de text
Automated code generation system for a compiler syntactic analysis phase based on free text processing

Rezumat teză de doctorat

Conducător științific:
Prof. Dr. Ing. Valentin SGÂRCIU

2011

Cuprins

1. INTODUCERE
1.1. Contextul texei

Tematica tezei se regăsește la intersecția a trei mari domenii: extragere de informații din texte structurate sau semi-structurate, data mining și generare automată de cod pentru diverse instrumente software. Primele două domenii – extragerea de informații și data mining fac parte din domeniul vast al prelucrării limbajului natural care se ocupă de analiza și întelegerea informațiilor descrise în limbaj natural. Generarea automată de cod este un concept care are în vedere obiectivul de a permite programatorului să se concentreze la logica din spatele programului mai mult decât la implementarea unor zone de cod care sunt generabile pe baza unor șabloane. Fiecare din cele trei domenii au implicat de-a lungul timpului interes în zona cercetării, existând o inter-dependență între ele.

Extragerea de informații (IE – Information Extraction) este procesul prin care se selectează structuri și se combină informațiile găsite în unul sau mai multe texte de intrare. Explozia de informații din mediul online a condus la o creștere a cererii de procesare și analiză a unor volume mari de date. Extragerea de informații reprezintă o categorie de procesare de documente care capturează informații specifice dintr-un document.

Diferența între procesul numit **Regăsire de informații (IR – Information Retrieval)** și IE constă în faptul că IR identifică un subset de *documente* dintr-o librărie de documente pe baza unor cerințe, pe când IE identifică un subset de *informații* în cadrul unui document. Datorită acestei diferențe majore între cele două concepte, în cadrul tezei am folosit tehnica IE, deoarece scopul final este generarea automată a codului sursă pentru un analizor sintactic pornind de la un document de intrare ce reprezintă tutorialul pentru limbajul descris în gramatica parserului.

Arhitectura unui sistem IE cuprinde o serie de module (sau componente) care procesează textul de intrare prin aplicarea regulilor impuse (Hobbs, 1994). Sistemul procesează textul creând structuri de date pentru secțiunile relevante ale intrării și eliminând secțiunile irelevante.

Data mining (Hand et al., 2001) (Tan et al., 2005) reprezintă procesul de analiză a datelor de intrare din diverse perspective și sumarizarea lor și a relațiilor dintre ele în informații utile. Ca obiectiv principal al procesului de **data mining** este găsirea de corelații sau șabloane între elemente din seturi foarte mari de date.

Pricipalele componente ale sale sunt:
- extragerea, procesarea și încărcarea datelor în sisteme complexe de stocare
- stocarea și gestiunea datelor în sisteme de baze de date multi-dimensionale
- analiza datelor prin metode statistice, prin metode bazate pe rețele neuronale sau prin antrenare
- prezentarea datelor într-un format util

Principalele categorii de activități implicate sunt: (Murty et al., 1999)
- **Clasificarea** - datele stocate sunt împărțite în grupuri predefinite
- **Clusterizarea** – datele sunt grupate conform cu relațiile logice care le caracterizează fără a cunoaște detalii despre aceste date (Marmanis et al., 2009) (Murty et al., 1999)
- **Asocieri –** identificarea relațiilor dintre variabile
- **Șabloane secvențiale –** datele sunt folosite pentru a anticipa tendințe și șabloane

Există mai multe nivele de analiză:

- **Rețele neuronale –** modele predictive non-liniare care sunt antrenate și reasamblează rețele neuronale biologice în structuri de date
- **Algoritmi genetici** – tehnici de optimizare care folosesc procese de tip combinații genetice, mutații și selecție naturală în proiectarea bazată pe conceptele evoluției naturale
- **Arbori de decizie –** arbori de structuri care reprezintă seturi de decizii. Aceste decizii generează reguli pentru clasificarea datelor de intrare. Astfel de metode includ *Arbori de Clasificare și Regresie (CART)* (Yohannes et al., 1999) , arbori de decizie CHAID (*Chi Square Automatic Interaction Detection*) (Neville, 1999)
- **Metoda celui mai apropiat vecin** – clasifică fiecare înregistrare pe baza unei combinații de clase de câte k înregistrări cele mai similare cu ea
- **Reguli induse –** extragere de reguli de tip *if-then* pe bază de metode statistice

Cel de-al treilea domeniu – **generarea automată de cod –** a avut parte de diverse abordări în zona compilatoarelor. (Aho et al., 1989) (Proebsting, 1995) Deși marea majoritate a generatoarelor de cod folosesc algoritmi bazați pe selectarea instrucțiunii, Aho și Proebsting propun metode bazate pe rescrierea regulilor care asociază un arbore șablon cu fiecare instrucțiune a mașinii targetate.

Una din cele mai de început metode de generare de cod bazate pe parsare de tip LR(1) *(Left-to-right-canonical-derivation)* a fost dezvoltată de către Glanville și Graham. (Glanville, 1977) (Glanville et al., 1978) Atunci când o regulă rulează se execută secvența cod mașină corespunzătoare acțiunii semantice implementate. Problema principală dezvăluită a fost legată de ambiguitățile de tip *shift-reduce* și *reduce-reduce* din timpul parsării LR. Ganapathi (Ganapathi, 1980) și Fischer (Ganapathi et al., 1982) au extins abordarea lui Graham-Glanville prin folosirea unui atribut al gramaticii care permite specificarea de predicate sintactice pentru situațiile ambigue.

Generatorul de cod ar trebui privit ca un ajutor prețios care înglobează experiența mai multor dezvoltatori, produce cod sursă consistent în câteva minute față de câteva luni – perioadă estimată pentru scrierea bucăților respective de cod – și conține un set mult mai mic de probleme decât cele introduse la scrierea de cod sursă pornind de la nimic. Totodată trebuie avut în vedere că un cod generat nu implică un succes garantat al proiectului, încă existând necesitatea de oameni de specialitate care să proiecteze și să acopere toate zonele goale din codul generat automat.

1.2. Obiectivele și conținutul lucrării

Scopul tezei este reprezentat de obținerea în mod automat a regulilor sintactice și lexicale necesare unui analizor sintactic pornind de la un document de intrare scris în limbaj natural care descrie specificațiile limbajului de programare pentru care se dorește crearea analizorului sintactic. Teza urmează două mari direcții; prima presupune identificarea și extragerea din cadrul documentului de intrare a secțiunilor care descriu sintaxa diverselor structuri sintactice și lexicale – pe baza acestora creându-se în mod automat regulile sintactice și lexicale din cadrul analizorului sintactic. Cea de-a doua direcție este reprezentată de generatorul automat de cod care interpretează rezultatele identificate la prima etapă și elaborează reguli sintactice și lexicale conforme cu un compilator de parsere ales.

Punctul de plecare al lucrării a fost direcția de cercetare urmată de Aho (Aho et al., 1989) în crearea de generatoare de cod automat pentru partea de *back-end* a compilatoarelor pornind de la specificații declarative care mapează arborele de reprezentări intermediare în cod mașină. Literatura de specialitate dezvăluie diverse articole în această direcție – (Proebsting, 1995), (Ganapathi, 1980), (Böhm, 2007).

Un compilator este o aplicație care acceptă un cod sursă scris într-un limbaj de programare cum ar fi de exemplu Java, C etc. și generează codul mașină corespunzător pentru o anumită arhitectură de mașini. Pentru a obține portabilitate se folosește o reprezentare intermediară IR (*Intermediate Representation*), compilatorul devenind astfel o aplicație cu două componente majore: prima din ele – *front-end* – mapează codul sursă la reprezentarea intermediară IR, iar cea de a doua numită *front-end* mapează reprezentarea IR la codul mașină. Astfel partea de *front-end* poate fi scrisă o singură dată, iar partea de *front-end* se modifică/adaptează la ce arhitectură a mașinii este nevoie. Fiecare din cele două componente ale compilatorului conțin o serie de sub-etape. (Fegaras, 2005)

Componenta de *front-end* conține următoarele sub-etape:

- *scanarea*: caracterele sunt grupate în unități atomice numite *tokeni*
- *parsarea*: un analizor recunoaște secvențe de astfel de *tokeni* în conformitate cu regulile unei gramatici și generează arborele de sintaxă AST (*Abstract Syntax Tree*)
- *analiza semantică*: realizează analiza tipurilor de date (verifică faptul că variabilele, funcțiile etc. din programul sursă sunt consistente cu definițiile lor și cu specificațiile limbajului) și translatează arborele AST la IR (reprezentarea intermediară)
- *optimizarea*: optimizează reprezentarea intermediară a codului sursă

Componenta de back-*end* conține următoarele sub-etape:

- *selectarea instrucțiunii*: mapează reprezentarea intermediară IR în cod asamblare
- *optimizarea codului*: optimizează codul asamblare folosind diverse tehnici – analiza fluxului de date, alocarea regișrilor etc.
- *emisia de cod*: generează codul mașină pentru codul de asamblare obținut

Inovația adusă de subiectul tezei constă în încercarea de automatizare a unei alte etape din cadrul unui compilator față de etapa codului intermediar propusă de Aho și Proebsting, și anume generarea automată a regulilor pentru scanare și parsare din componenta de *front-end*. Regulile sintactice și lexicale necesită o cunoaștere exactă a sintaxei descrisă de specificațiile limbajului de programare.

Pentru a extrage informațiile despre sintaxa diverselor structuri sintactice din cadrul specificațiilor limbajului de programare sistemul definit în cadrul lucrării propune ca și intrări un manual de referință pentru limbajul de programare luat în calcul împreună cu un lexicon al structurilor întâlnite în acel limbaj.

Sistemul care extrage informațiile despre sintaxa structurilor sintactice este un sistem de tip IE (*Information Extraction*) ce cuprinde o serie de patru module distincte ce acoperă următoarele secțiuni: comentariile limbajului de programare, cuvintele cheie sau rezervate ale acestuia, operatorii și structurile sintactice. Fiecare astfel de modul este implementat pe baza unui algoritm propus în cadrul tezei.

Pentru testatea algoritmilor definiți am folosit un set de cinci documente de intrare, fiecare

reprezentând un tutorial pentru următoarele limbaje de programare: **C++**, **Java**, **PHP**, **e** și **SystemVerilog**. Șabloanele utilizate în cadrul algoritmilor IE s-au dovedit a fi generatoare de mulțimi de date ce reprezintă bucăți din cadrul documentelor de intrare ce ar putea reprezenta sintaxa pentru structurile căutate; pentru determinarea rezultatului celui mai relevant am folosit metode de clusterizare de tip partiționare împreună cu o serie de parametrii propuși care indică o măsură a calității rezultatelor.

Generatorul de cod automat dezvoltat primește ca intrări rezultatele obținute cu algoritmii de extragere de informații și generează regulile sintactice și lexicale conform cu sintaxa ANTLR. Intrările pentru genratorul de cod sunt împachetate într-un limbaj comun, fiind astfel ușor de citit și de interpretat.

Pentru testarea arborelui de gramatică obținut prin apelul generatorul de cod automat am elaborat o serie de 135 de teste, grupate în două categorii majore: teste pozitive care reprezintă cod sursă valid și teste negative care reprezintă cod sursă invalid ce trebuie rejectat.

Platforma de testare implică și o secțiune de raportare a rezultatelor testelor. Este permisă o interogare configurabilă bazată pe tipul structurii sintactice. Interfața web oferă un rezumat al testelor afișând conținutul testului și rezultatul rulării acestuia.

2. PROCESARE AUTOMATĂ DE TEXT

2.1. Prezentare generală a procesării automate de text

Există o paletă largă de abordări care au în vedere tehnologiile bazate pe limbajul uman (HLT = *Human Language Technology*). Totuși, informația astfel definită nu poate fi descoperită folosind o reprezentare simplă de pachet de cuvinte. Entitățile referite într-un document, proprietățile și relațiile asertate între acestea nu pot fi identificate folosind o reprezentare de tip vector de spațiu. Deși înțelegerea completă a limbajului natural este încă departe de capacitățile tehnologiilor curente, metodele folosite de extragerea de informații (IE – *information extraction*) oferă mai multe acuratețe, fiind capabile să recunoască diverse tipuri de entități în text și câteva relații între acestea - cuprind procesarea de limbaj natural (NLP), recunoașterea de voce, generarea de text și extragerea de text (*Text Mining*).

În direcția procesării de limbaj natural s-au dezvoltat tehnologii care sunt inspirate din lingvistică – textul este parsat sintactic folosind informații descrise de o gramatică formală și un lexicon; informația rezultată este apoi interpretată semantic și folosită pentru a extrage informații în legătură cu subiectul textului respectiv.

NLP poate parsa în adâncime (ia în considerare fiecare parte de vorbire din fiecare propoziție și o analizează semnatic) sau poate parsa la un nivel mai superficieal (ia în considerare doar anumite pasaje sau fraze realizând o analiză semantică limitată). NLP poate folosi măsuri statistice pentru cuvinte ambigue sau pentru părți multiple din aceeași frază. Include tehnologii cum ar fi: găsirea cuvântului care rezultă prin eliminarea sufixelor (*word stemming*), gruparea frazelor alcătuite din mai multe cuvinte, normalizarea sinonimelor, etichetarea părților de vorbire (POS Tagger = *part of speech tagger*), a cuvintelor ambigue, determinarea rolului sintactic (de exemplu: subiect și obiect).

Tehnologia de extragere de text (*Text Mining*) reprezintă o abordare mai nouă și folosește metode din domeniul identificării de informații, statistică. Scopul său nu este să înțeleagă tot sau o parte mare din text ci să extragă șabloane dintr-un număr mare de documente. Cea mai simplă formă de *Text Mining*

este extragerea de informații. Alte forme includ clasificarea textului în mod automat, sumarizarea automată, extragerea de topicuri din text sau a grupuri de text și analiza acestora.

Cercetările din direcția *Data Mining* presupun că informația este deja mixată într-o bază de date relațională. Din păcate însă, pentru multe aplicații, infomația electronică disponibilă se află într-o formă de documente nestructurate. Astfel extragerea de text (*Text Mining*) – descoperirea de informații folositoare din text nestructurat - a devenit o componentă extrem de importantă. Cea mai mare parte din munca în domeniul *Text Mining* nu exploatează procesarea de limbaj natural, tratând documentele ca pe un pachet de cuvinte neordonate – așa cum se procedează în extragerea de informații. Un model de vector ce conține text reprezintă un vector care specifică frecvența pentru fiecare dintre cele mai frecvente cuvinte distincte care apar în corpus. O astfel de abordare s-a dovedit eficientă pentru un număr standard de acțiuni – cum ar fi extragerea de documente, clasificare, clusterizare. (Baeza-Yates et al., 1999)

2.2. Metodologia de extragere de informații

Cea mai mare parte a informației stocate în format digital este ascunsă în texte scrise în limbaj natural (NL – *natural language*). Descoperirea de informații (IR – *information retrieval*) ajută la localizarea documentelor care ar putea conține aspectele dorite, dar nu permite și posibilitatea de a primi răspuns la diverse cereri. Scopul metodei de extragere de informații (IE) este de a identifica bucățile de informații dorite din texte în limbaj natural și de a le stoca într-o formă care să permită interogarea și procesarea automată. Se definesc conceptele de *slot* și *șablon*. *Slotul* conține o singură bucată de informație (de exemplu un nume sau o adresă); *șablonul* conține o listă sau un tuplu de sloturi și/sau alte șabloane (de exemplu: o listă de adrese conține o listă de șabloane de tip adresă). Șabloanele pot fi aranjate în structuri ierarhice.

În mod formal o problemă de tip IE se definește prin datele de intrare și datele țintă. Intrarea poate fi reprezentată de documente scrise în limbaj natural sau de documente semi-structurate care pot apărea sau nu și pe Web - cum ar fi tabelele sau listele cu enumerări. Obiectivul extragerii pentru o problemă de tip IE poate fi o relație de k-tuple (unde k este numărul de atribute într-o înregistrare) sau poate fi un obiect complex dintr-o structură de date ierarhizată.

Programele care rezolvă probleme de tip IE sunt numite în literatura de specialitate *extractori*. Un astfel de extractor a fost inițial definit ca o componentă a unui sistem de integrare de informație care oferă o interfață unificată pentru accesarea a mai multor surse de informații. Sistemele de tip IE sunt instrumente software care sunt proiectate să genereze extractori. Un extractor realizează de obicei o procedură de potrivire cu unul sau mai multe șabloane.

Subiectul prezentei lucrări – extragerea de informații precise dintr-un tutorial se apropie de problemele de tip IE pentru mediul online deoarece procesează documente semi-structurate. Sistemele IE tradiționale se folosesc de tehnici de tip NLP cum ar fi lexicoane si gramatici, iar sistemele IE pentru Web folosesc tehnici de tip "*data mining*" pentru a exploata șabloane sintactice sau structuri bazate pe șabloane. Documentele tehnice care oferă un ghid pentru un limbaj de programare au o structură ușor de pus într-un șablon în ceea ce privește sintaxa codului sursă. Deși fiecare astfel de document poate avea un șablon diferit, acesta nu diferă cu mult.

În ultimii ani s-au propus diverse abordări pentru sistemele de tip IE care includ tehnici de învățare și de "*data mining*" cu diferite grade de automatizare. Cercetarea din ultima perioadă a fost inspirată de MUC – *Message Understanding Conferences*. Există cinci mari direcții definite pentru IE printre care construcția elementului șablon, definirea relației șablon și producerea de scenarii pentru șabloane. Contribuția semnificativă adusă de MUC a împărțit abordările IE în două clase diferite:

- **abordări de tip MUC** – AutoSolg (Riloff, 1993), LIEP (Huffman, 1996), PALKA (Kim et al., 1995), HASTEN (Krupka, 1995), CRYSTAL (Slonnenger et al., 1995)
- **abordări de tip Post-MUC** – WHISK (Soderland, 1999), RAPIER (Califf et al., 1998), SRV (Freitag, 1998), WIEN (Kushmerick et al., 1998), SoftMealy (Hsu, 1998), STALKER (Muslea et al., 1999)

Sistemele de tip IE pot fi analizate din trei perspective: dificultatea problemei, tehnologiile folosite și evaluarea atât a efortului depus de utilizator în procesul de învățare cât și a necesității portării sistemului pe diferite domenii. Se poate observa că gradul de ușurință a întelegerii textului de către o mașină crește cu nivelul de structurare al documentului. Documentele de intrare semi-structurate sunt cele care au o structură cât de cât regulată și datele pot fi aranjate în format HTML sau non-HTML.

3. PROIECTARE ANALIZOARE SINTACTICE

3.1. Prezentare generală a principalelor concepte din domeniul compilatoarelor

Definiție: *Parsarea* este descrisă ca fiind procesul de structurare a unei reprezentări liniare în concordanță cu o gramatică dată.

Forma abstractă a definiției prezentate se datorează posibilităților numeroase de interpretare pe care le permite. Reprezentarea liniară poate fi considerată o propoziție, un program, un șablon etc. deci de fapt orice secvență liniară în care elementele precendente restricționează în vreun fel elementul următor. În contextul unui compilator/translator, parsarea (sau analiza sintactică) este o componentă din etapa numită *front-end* și se ocupă cu gruparea părților constituente ale sursei unui program în construcții gramaticale.

Definiție: Un *compilator* este o aplicație care citeste un program scris într-un limbaj – limbajul sursă - și îl translatează într-un program echivalent din alt limbaj – limbajul destinație. În procesul de translatare compilatorul raportează utilizatorului prezența erorilor în programul sursă. (Aho et al., 1986)

Definiție: *Analiza liniară* reprezintă gruparea fluxului de caractere ce constituie programul sursă citit de la stânga către dreapta în unități unice care au un sens colectiv, numite *tokeni*.

Definiție: În contextul unui compilator analiza liniară poarta numele de *analiză lexicală* sau *scanare*.

Definiție: *Parsarea* sau *analiza sintactică* este o analiza ierarhică ce implica gruparea *tokenilor* din programul sursă în reguli gramaticale.

Definiție: *Analiza semantică* se ocupă de verificarea programului sursă în vederea erorilor semantice. Analiza semantică se ocupă de asemenea cu colectarea de informatii despre tipurile de date pentru fazele următoare. Folosește structura ierarhică determinată la faza analizei sintactice pentru a identifica

operatori și operanzi ai expresiilor. (Aho et al., 1986)

Definiție: *Tabela de simboluri* este o structură de date ce contine câte o înregistrare pentru fiecare identificator împreună cu câmpuri ce descriu atributele acestuia. Aceste atribute stochează informații despre spațiul alocat pentru acel identificator, tipul său, scopul său (locația în program în care este valid) iar în cazul funcțiilor și procedurilor - numărul și tipul argumentelor, tipul returnat etc.

Definiție: *Detecția erorilor și raportarea acestora* se întâlnește în oricare dintre etapele unui compilator. Cea mai mare parte din erori e reprezentată de cele detectate de fazele de analiza lexicală, sintactică și semnatică.

Definiție: *Generarea de cod intermediar* este faza în care unele compilatoare generează o reprezentare intermediară explicită a programului sursă care este ușor de realizat și de asemenea usor de translatat în programul destinație.

Definiție: *Optimizarea de cod* este faza în care se încearcă îmbunătățirea codului intermediar.

Definiție: *Generarea de cod* se ocupă cu generarea codului destinație care este de obicei cod masină sau limbaj de asamblare.

Fazele unui compilator sunt grupate în două mari categorii – partea de *front-end* și cea de *back-end*. Partea de *front-end* cuprinde etapele care depind în principal de limbajul sursă și sunt independente de masina țintă. Aceste etape includ analiza lexicală și analiza sintactică, crearea tabelei de simboluri, analiza semantică și generarea de cod intermediar. Partea de *back-end* include porțiunile din compilator care depind de masina țintă. Tot în partea de *back-end* se găsesc anumite aspecte legate de optimizare, de generarea de cod, de gestionarea erorilor și operații cu tabela de simboluri.

Analizorul lexical este responsabil și cu câteva aspecte foarte folositoare la nivel de interfață utilizator:

- eliminarea comentariilor și a spatiilor (blank, tab și caractere de început de linie nouă) din programul sursă
- corelarea mesajelor de eroare din compilator cu programul sursă (analizorul lexical memorează numărul liniilor pentru tokenii găsiți astfel încât atunci când este întâlnită o eroare să se trimită utilizatorului și informația ajutătoare de tip eroarea este la linia și coloana)
- în unele compilatoare analizorul lexical face o copie a programului sursă cu mesajele de eroare corespunzătoare
- dacă limbajul scanat suportă macro-uri de preprocesare acestea se rezolvă tot la nivelul analizorului lexical

Parserul (analizorul sintactic) primește un șir de tokeni de la analizorul lexical și verifică dacă șirul poate fi generat de regulile gramaticii.

4. TRANSLATARE SEMI-AUTOMATĂ A DOCUMENTELOR LA ANALIZOARE SINTACTICE

4.1. Proiectare sistem IE

Extragerea de informații din documentele de intrare se face pe baza unui Sistem de tip IE (Information Extraction) (Hobbs, 1994) care conține un set de algoritmi propuși. Gradul de generalitate al algoritmilor este dat de posibilitatea de a fi rulați pe orice document care se încadrează în clasa de documente semi-structurate a tutorialelor.

Sistemul IE propus în cadrul tezei (Fig. 1.) are următoarele caracteristici:

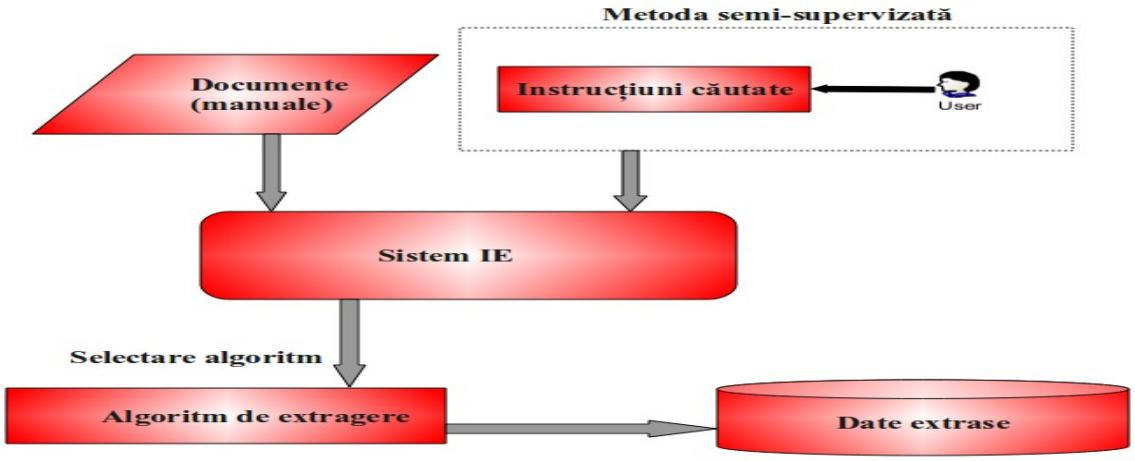

Fig. 1. Fluxul de operații în cadrul sistemului IE proiectat

Metoda IE folosită: sistemul folosește o metodă IE de tip semi-supervizat prin care se primește un șablon general pe care se construiește expresia regulată.

Abordarea este de tip "*top-down*" pornindu-se de la o expresie regulată cât mai generală pe rezultatul căreia se aplică termeni specifici.

Algoritmul de învățare este definit prin extragere de șablon.

Șablon este numele generic pentru o instrucțiune din limbajul de programare studiat; nu conține sub-șabloane care să poată fi regăsite cu ușurință în textul dat

Tokenizarea se face la nivel de cuvânt.

Tipul regulii de extragere se descrie printr-o expresie regulată împreună cu un orizont de text care să acopere o zonă de tokeni aflați înaintea textului țintă și o zonă de tokeni aflați după textul ținta, adică se ia în calcul și contextul în care apare textul țintă.

Sistemul IE definit primeste la intrare documente tehnice reprezentând tutoriale ale unor limbaje de programare, acestea încadrându-se în clasa de documente semi-structurate. Informațiile dorite din aceste documente sunt structurile de sintaxă pentru diverse construcții din cadrul limbajului de programare, atât construcții lexicale cât și sintactice. Aceste informații sunt definite în raport cu regulile

lexicale și regulile sintactice de care are nevoie un analizor sintactic (parser) (Fig. 2.)

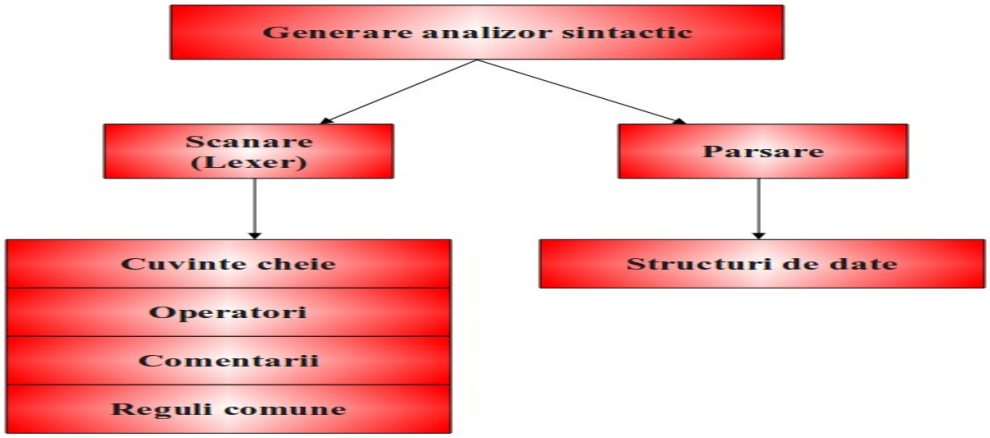

Fig. 2. Componente analizor sintactic

Sistemul IE conține un set de patru algoritmi de extragere de informații, câte unul pentru fiecare clasă de structuri întâlnită într-un limbaj de programare clasic. Algoritmii IE propuși în cadrul tezei acoperă următoarele clase de structuri sintactice și lexicale:

- Structuri lexicale: cuvinte cheie; operatori și precedența acestora; comentarii
- Structuri sintactice

Comentariile într-un limbaj de programare se regăsesc de obicei într-o secțiune separată a tutorialului. Pe baza unui set de teste al cărui scop a fost să determine aria corespunzătoare comentariilor în tuturial s-a observat că șablonul "*comment*" este întâlnit în multe locuri, fiind nevoie de o etapă de determinare a celei mai relevante informații extrase despre comentarii.

Pentru a identifica această zonă relevantă de informație am plecat de la ideea că un limbaj de programare permite în general comentariile de tip linie – comentarii aflate pe aceeași linie cu un cod sursă valid. Identificând orizontul de text referitor la comentariile linie, pe baza unei estimări am obținut o fereastră de text care poate conține toate comentariile descrise în documentul de intrare, atât cele linie, cât și cele bloc.

Algoritm de identificare a comentariilor

Pas 0. – Încărcare fișier de intrare

- se încarcă tutorialul într-o structură de date adaptată la limbajul de programare în care se implementează algoritmul (în cazul nostru: limbajul Perl (Allen, 2011))

Pas 1. - Identificare zonă în care apare șablonul dorit

- se identifică fraza necesară șablonului de căutare – comentariile puse pe aceeași linie cu codul sursă
- se parcurge documentul până se întâlnește fraza propusă
- se determină un orizont de text care să cuprindă textul identificat plus o expandare la o fereastră de text pe baza unei estimări propuse

Pas 2. - Extragerea marcatorilor pentru comentarii

- se caută caractere care ar putea aparține setului de caractere alocat comentariilor

- se elimina caracterele alfa-numerice
- se face o curățare a textului obținut
- se scrie un fișier text cu posibilele caractere alocate comentariilor

Cuvintele cheie (rezervate) ale unui limbaj de programare se regăsesc de obicei în cadrul unui manual ce descrie un limbaj de programare grupate într-o singură secțiune pentru o mai ușoară înțelegere a documentului de către utilizator și pentru ca acesta să evite să le folosească în contexte care vor conduce la erori de sintaxă.Secțiunea în care sunt definite cuvintele cheie conține de cele mai multe ori o modalitate de separare a acestora, fie printr-un tabel, fie printr-o listă în care acestea sunt despărțite de un caracter non-alfanumeric etc.

Algoritm de identificare a cuvintelor cheie

Pas 0. – Încărcare fișier de intrare
- se încarcă tutorialul într-o structură de date adaptată la limbajul de programare în care se implementează algoritmul (în cazul nostru: limbajul Perl (Allen, 2011))

Pas 1. - Identificare zonă în care apare șablonul dorit
- se identifică șablonul de căutare
- se parcurge documentul până se întâlnește fraza propusă
- se determină un orizont de text care să cuprindă textul identificat plus o expandare la o fereastră de text pe baza unei estimări propuse

Pas 2. - Extragerea cuvintelor cheie
- rezultatele obținute se clusterizează folosind algoritmul de clusterizare bazat pe partiționare K-means
- pentru fiecare cluster găsit se fac următoarele prelucrări
  - se determină caracterele separator cu frecvența cea mai mare
  - se face o curățare a clusterilor:
    - clusterii cu număr foarte mic de linii sunt eliminați
    - clusterii care conțin caractere speciale sunt eliminați
- se face o analiză a rezultatelor obținute în urma clusterizării pornind de la o serie de parametrii propuși în cadrul tezei

Algoritm de identificare a operatorilor

Pas. 0. – Încărcare fișier de intrare
- se încarcă tutorialul într-o structură de date adaptată la limbajul de programare în care se implementează algoritmul (în cazul nostru: limbajul Perl (Allen, 2011))

Pas 1. - Identificare capitol ce conține șablonul dorit
- se identifică fraza necesară șablonului de căutare
- se parcurge documentul până se găsește fraza propusă
- se determină capitolul în care se află fraza

Pas 2. - Identificare zonă în care apare șablonul dorit
- se face o aproximare a numărului de caractere care pot să aibă legătura cu fraza dată
- se delimitează un orizont pe baza acestei aproximări

Pas 3. - Extragerea operatorilor
- rezultatele obținute se clusterizează folosind algoritmul de clusterizare bazat pe partiționare K-means

- pentru fiecare cluster găsit se fac următoarele prelucrări
  - se caută acele caractere care pot reprezenta un operator
  - se elimina alfa-numerice
  - se face o curățare a textului obținut
  - se scrie un fișier text cu operatorii identificați
- se face o analiză a rezultatelor obținute în urma clusterizării pornind de la o serie de parametrii propuși în cadrul tezei

Algoritm de identificare a structurilor de date

Pas 0. – Încărcare fișier de intrare
  - se încarcă tutorialul într-o structură de datea daptată la limbajul de programare în care se implementează algoritmul (în cazul nostru: limbajul Perl (Allen, 2011))

Pas 1. - Identificare zonă în care apare șablonul dorit
- se identifică șablonului de căutare
- se parcurge documentul până se găsește șablonul căutat, reținându-se toate aparițiile sale și apoi determinând apariția cea mai relevantă
- se determină un orizont de text care să cuprindă textul identificat plus o expandare la un număr de linii de mai sus și de mai jos

Pas 2. - Extragerea sintaxei
- se caută în orizontul de text identificat caractere care ar putea aparține setului de caractere speciale care ar putea să marcheze sintaxa unei construcții validând astfel paragraful ca fiind cel căutat.
  - Caracterele speciale căutate includ:
    - [ ] - marchează în general parametri opționali într-o sintaxă
    - { } - marchează în general delimitări de blocuri
    - ; - marchează în general sfârșitul unei instrucțiuni
- se formatează rezultatul obținut astfel:
  - caracterele speciale identificate se translatează la formatul necesar gramaticii pentru analizorul sintactic
  - se identifică dacă există cuvinte rezervate pe baza rezultatelor obținute cu algoritmul de identificare a cuvintelor cheie prezentat anterior
  - se identifică acele cuvinte care se încadrează în clasa identificatorilor pentru un limbaj de programare și se notează toate cu același simbol

4.2. Rezultate experimentale ale sistemului IE

Algoritmii propuși au fost aplicați pe următoarele documente de intrare:
1) Draft Standard for the Functional Verification Language e (Design Automation Standard Committee, 2007)
2) C++ Language Tutorial (Soulie, 2007)
3) PHP Tutorial From beginner to master (PHP Community)
4) The JavaTM Language Specification Third Edition (Gosling, 2005)
5) SystemVerilog 3.1a Language Reference Manual (Accellera Organization, 2004)

4.2.1. Extragere de informații despre comentarii

Observații:

- textul nu poate conține două comentarii de tip linie fără să existe câte o linie separată pentru fiecare
- algoritmul propus găsește ca și marcator de comentariu linie caracterul “;”. Acest caracter este folosit în general ca și marcator de sfârșit de instrucțiune. Generatorul pentru regulile lexicale se bazează pe această observație și nu îl introduce în lista posibililor marcatori de comentariu linie
- rezultatele obținute în urma aplicării algoritmului propus arată toți marcatorii de comentarii găsiți. Generatorul de reguli lexicale face o filtrare a acestor marcatori astfel încât să se ia în considerare doar marcatorii unici

Rezultate:

- un sumar al rezultatelor vs. ceea ce ne așteptăm să obținem este prezentat în Tabel 1.

Limbaj de programare	***Tipuri de comentarii***	***Comentarii detectate***	***Rata de detecție***
Draft Standard for the Functional Verification Language e	3	2	66.66
SystemVerilog 3.1a Language Reference Manual	2	2	100
The JavaTM Language Specification Third Edition	4	4	100
PHP Tutorial From beginner to master	2	2	100
C++ Language Tutorial	2	2	100

Tabel 1. Rezultate experimentale pentru dectecția comentariilor

4.2.2. Extragere de informații despre cuvintele cheie

Algoritmul propus de extragere de informații despre cuvintele cheie a generat câte un set de rezultate pentru fiecare document primit la intrare, așa cum se poate observa în tabelul următor:

Document	***Număr zone de text identificate***
Draft Standard for the Functional Verification Language e	39
C++ Language Tutorial	43
The JavaTM Language Specification Third Edition	83
SystemVerilog 3.1a Language Reference Manual	119

Tabel 2. Zonele de text identificate de algoritmul de extragere a cuvintelor cheie

Rezultatele obținute au fost trecute printr-un algoritm de clusterizare care a avut drept scop determinarea informațiilor celor mai relevante, micșorând aria de căutare pe baza similiarității acestora.

(Kaufman et al., 1990)

- rezultatele de mai sus au fost clusterizate folosind algoritmul de clusterizare K-means (MacQueen, 1967) (Hurtigan et al., 1989) (Dubes et al., 1988)
- numărul clusterilor (k) a fost ales ca în Tabel 3. pe baza unui set de teste experimentale
- distanța folosită de algoritmul K-means a fost aleasă ca fiind Distanța Euclidiană: (Wolfram Mathematica Documentation Center, 2011)

$$d(i,j)=\sqrt{(|x_{i1}-x_{j1}|^2+|x_{i2}-x_{j2}|^2+\ldots+|x_{ip}-x_{jp}|^2)}$$

- din Tabel 2. se poate trage concluzia că șablonul folosit pentru extragerea cuvintelor cheie este unul foarte permisiv pentru documentul "The JavaTM Language Specification Third Edition", rezultatele obținute fiind în număr foarte mare

Document	***Parametrul k din cadrul algoritmului K-means***
Draft Standard for the Functional Verification Language e	6
C++ Language Tutorial	6
The JavaTM Language Specification Third Edition	11
SystemVerilog 3.1a Language Reference Manual	9

Tabel 3. Alegerea parametrului k din algoritmul K-means pentru cuvintele cheie

4.2.3. Extragere de informații despre operatori

Algoritmul de extragere de informații propus pentru identificarea operatorilor a primit două șabloane de căutare:

- primul din ele e mai strict căutând rezultate care să descrie precedența operatorilor
  - rezultatele obținute în acest caz sunt foarte bune pentru documentul "*Draft Standard for the Functional Verification Language e*", dar foarte slabe în cazul celorlalte documente de intrare
  - deși am obținut rezultate foarte bune în cazul primului document de intrare folosind un șablon mai strict, vom aplica șablonul mai relaxat și în cazul acestui document pentru a putea face comparații între rezultate
- cel de-al doilea șablon este mai relaxat, căutând informații despre orice fel de operatori
  - rezultatele prezentate în continuare sunt cele obținute în urma aplicării șablonului de extragere mai relaxat

Algoritmul a generat câte un set de rezultate pentru fiecare document primit la intrare, așa cum se poate observa în tabelul următor. Numărul de rezultate este destul de mare în fiecare din cele patru rezultate deoarece șablonul de extragere este extrem de permisiv

Document	*Număr zone de text identificate*
Draft Standard for the Functional Verification Language e	233
C++ Language Tutorial	166
The JavaTM Language Specification Third Edition	440
SystemVerilog 3.1a Language Reference Manual	370
PHP Tutorial From beginner to master	18

Tabel 4. Zonele de text identificate de algoritmul de extragere a operatorilor

Rezultatele obținute au fost trecute printr-un algoritm de clusterizare care a avut drept scop determinarea informațiilor celor mai relevante, micșorând aria de căutare pe baza similiarității acestora. (Kaufman et al., 1990)

- rezultatele de mai sus au fost clusterizate folosind algoritmul de clusterizare K-means (MacQueen, 1967) (Hurtigan et al., 1989) (Dubes et al., 1988)
- numărul clusterilor (k) a fost ales ca în Tabel 5. pe baza unui set de teste experimentale
- distanța folosită de algoritmul K-means a fost aleasă ca fiind Distanța Euclidiană: (Wolfram Mathematica Documentation Center, 2011)

$$d(i,j)=\sqrt{(|x_{i1}-x_{j1}|^2+|x_{i2}-x_{j2}|^2+\ldots+|x_{ip}-x_{jp}|^2)}$$

Document	*Parametrul k din cadrul algoritmului K-means*
Draft Standard for the Functional Verification Language e	9
C++ Language Tutorial	8
The JavaTM Language Specification Third Edition	7
SystemVerilog 3.1a Language Reference Manual	8
PHP Tutorial From beginner to master	3

Tabel 5. Alegerea parametrului k din algoritmul K-means pentru operatori

4.2.4. Indicatori de calitate pentru rezultatele clusterizării

Analiza rezultatelor clusterizarii presupune definirea unor indici de calitate în raport cu rezultatele obținute și extragerea unor concluzii legate de relevanța clusterilor pe baza acestora. (Raskuti et al., 1999) (Dubes et al., 1979) Pentru subiectul tezei, propunem următoarele clase de **indicatori de calitate** pentru rezultatele obținute în urma clusterizării:

a) Densitate clusteri

Această analiză pune în evidență frecvența cu care apare șablonul căutat în zona de text identificată și cât de bine a fost aleasă fereastra de text în căutări. Concluziile identificate sunt:

- cu cât clusterul se regăsește într-un interval mai larg cu atât similaritatea între membrii

clusterului scade

- dacă intervalul este mic și numărul de clusteri este mare atunci similaritatea membrilor clusterilor este ridicată

b) Relevanță clusteri în raport cu pozițiile centroiziilor față de valorile medii ale observațiilor

Definim **Centroidul** unui cluster ca fiind punctul în care valoarea parametrului este dată de media valorilor parametrilor ale tuturor punctelor din cluster: (Ye, 2007)

$$c_j = \frac{1}{n_j} \cdot \sum_{v \in p_j} x_v$$

unde n_j este numărul de elemente din p_j

În urma calculului centroidului se observă unele diferențe între rezultatele obținute la centroid și mediile punctelor din cadrul componentelor clusterilor, ceea ce implică următoarea concluzie: cu cât diferența între cele două valori este mai mare cu atât clusterul respectiv conține și puncte care nu sunt relevante și care pot fi eliminate. (Fig. 3.) (Fig 4.)

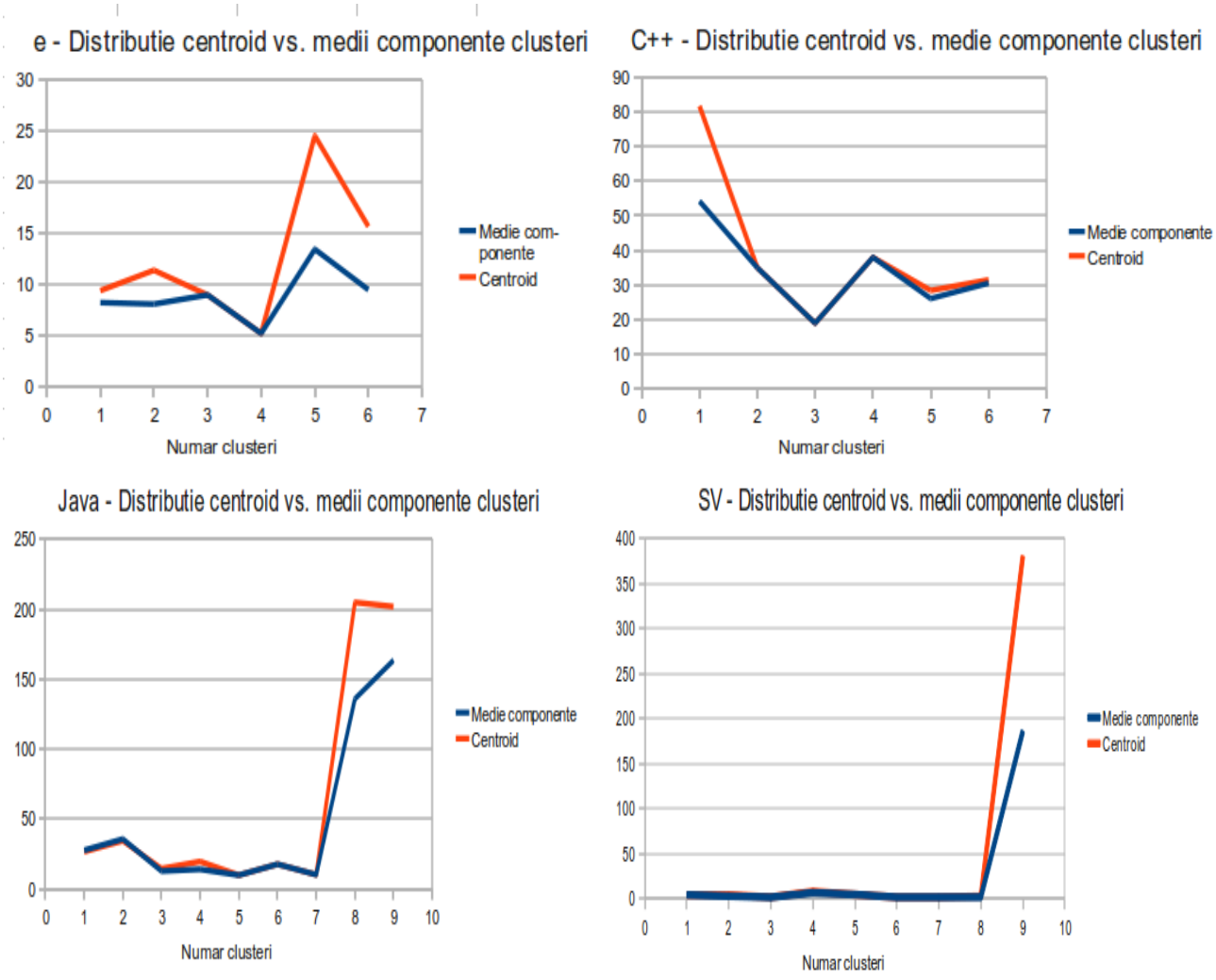

Fig. 3. Distribuție centroizi vs. medii componente clusteri (cuvinte cheie)

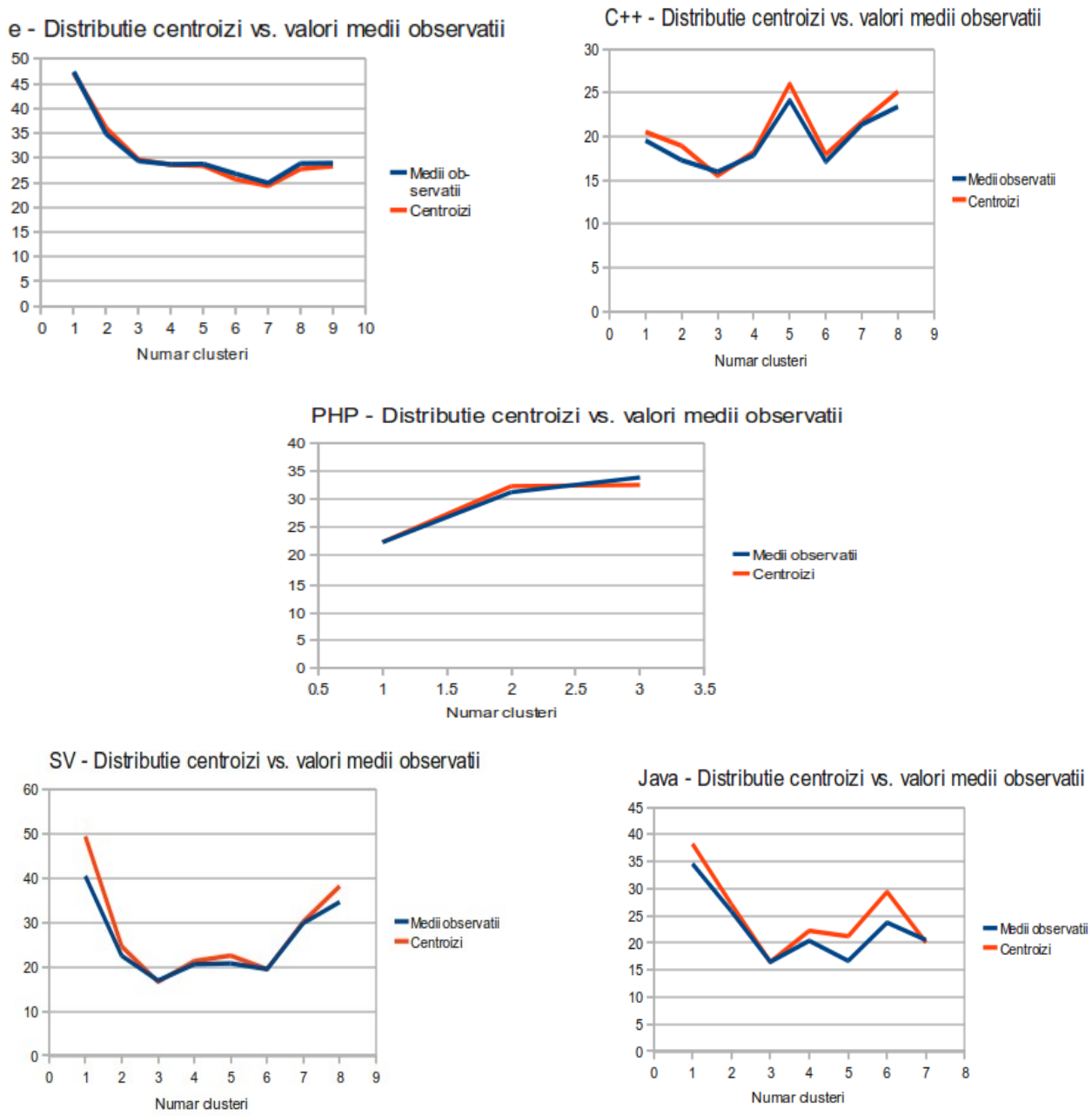

Fig. 4. Distribuție centroizi vs. medii componente clusteri (operatori)

c) Dimensiune clusteri

Un alt parametru care influențează calculul factorului de calitate al algoritmului este reprezentat de **dimensiunea** clusterilor:

- cu cât dimensiunea este mai apropiată de un prag mediu cu atât factorul de calitate crește
- dimeniunile foarte mici sau cel foarte mari conduc la un factor de calitate în scadere

Propunem următoarea codificare pentru parametrul dimensiune cluster conform cu Tabel 6. (pentru cuvintele cheie) și Tabel 7. (pentru operatori)

Dimeniune cluster	*Pondere – cuvinte cheie*
[1 .. 2]	0.10
[3 .. 7]	0.40
[7 .. 10]	0.80
[10 .. 15]	0.50
[15 .. 30]	0.30
[30 .. 50]	0.10

Tabel 6. Ponderi dimensiune cluster cuvinte cheie

Dimeniune cluster	*Pondere – operatori*
[1 .. 4]	0.10
[4 .. 10]	0.15
[10 .. 20]	0.30
[20 .. 30]	0.80
[30 .. 40]	0.60
[40 .. 80]	0.10

Tabel 7. Ponderi dimensiune cluster operatori

d) Relevanța clusterului în cadrul rezultatelor

Propunem următorul indicator ce reflectă relevanța clusterului în cadrul rezultatelor, indicator pe care îl denumim *REL.* Formula propusă este:

$$REL_{c_j} = \frac{1}{k} \cdot \sum_{i=1}^{m} x_i * P_i, j = 1 .. N$$

unde:
N = număr clusteri
k = numărul de puncte dintr-un cluster
x = observațiile definite ca fiind numărul de linii din fiecare punct al unui cluster
P = ponderile cuvintelor cheie/operatorilor (Tabel 6., Tabel 7.)

e) Relevanța algoritmului

Propunem următorul indicator (pe care îl denumim *SCOR)* care reflectă relevanța algoritmului aplicat pe fiecare document de intrare. Formula indicatorului am definit-o astfel:

$$SCOR = \frac{1}{N} \cdot \sum_{i=1}^{N} |max_{x_i} - y_i| * P_i * \left\{ 1 - \frac{1}{D_i} \right\}$$

unde:
N = număr clusteri
y = mediile observațiile x din formula anterioară
x = observațiile definite ca fiind numărul de cuvinte din fiecare element al unui cluster

P = ponderile cuvintelor cheie (Tabel 4.5.)
D = ponderile dimensiunilor clusterilor (Tabel 4.7.)

Pentru a determina clusterul cel mai relevant în raport cu probabilitatea de a conține operatorii identificați în cadrul documentului de intrare, luăm în considerare parametrii definiți anterior (împreună cu ponderile propuse în cadrul tezei), și anume:

- dimensiunea clusterilor
- densitatea clusterilor (gradul de împrăștiere a elementelor în cadrul clusterului)
- distribuția centroizilor față de valorile medii ale componentelor
- relevanța clusterului – distanța propusă în cadrul tezei – extremele în cazul distanței definite indică ori un număr de cuvinte cheie foarte mic, ori un număr de cuvinte cheie foarte mare

Analiza rezultatelor clusterizării pentru cuvintele cheie:

a) ***Draft Standard for the Functional Verification Language e***

- clusterii cu dimensiune foarte mică sunt: $dimC_3 = 2$
- clusterii cu gradul de împrăștiere cel mai mare sunt: C_5 și C_6 - de peste 6000 de unități
- clusterii care au valori de extrem ori foarte mari ori foarte mici pentru relevanță sunt: C_1 și C_4 (valori foarte mici), C_5 (valoare foarte mare) (Tabel 4.8.)
- distibuția centrozilor față de media componentelor clusterilor indică o valoare de extrem pentru clusterul C_5
- clusterul cel mai relevant devine C_2

b) ***C++ Language Tutorial***

- clusterii cu dimensiune foarte mică sunt: $dimC_3 = dimC_4 = 1$
- clusterii cu gradul de împrăștiere cel mai mare sunt: C_5 și C_6 - de peste 1000 de unități
- clusterul cu relevanța cea mai mică este C_3
- clusterii cel mai relevanți sunt C_1 și C_2 - îl luam în calcul și pe C_1 pentru că are o densitate foarte bună, deși reprezintă un extrem pentru distribuția centroizilor față de mediile componenetelor clusterilor

c) ***The JavaTM Language Specification Third Edition***

- clusterii cu dimensiune foarte mică sunt: C_5, C_6, C_9
- clusterii cu gradul de împrăștiere cel mai mare sunt: C_2 și C_8 - de peste 5000 de unități
- clusterii care au valori de extrem foarte mici pentru relevanță sunt: C_3, C_4, C_5, C_7
- distibuția centrozilor față de media componentelor clusterilor indică o valoare de extrem pentru clusterul C_8
- clusterul cel mai relevant devine C_1

d) ***SystemVerilog 3.1a Language Reference Manual***

- clusterii cu dimensiune foarte mică sunt: $dimC_1, C_4, C_5, C_6, C_7 < 10$, restul având o dimensiune medie de aproximativ 20
- valorile centroizilor și valorile medii ale componentelor sunt extrem de mici în aproape toți clusterii (până în 10), singurul cluster cu o valoare mai ridicată fiind C_9

- clusterul cel mai relevant devine C_9

Analiza rezultatelor clusterizării pentru operatori:

a) ***Draft Standard for the Functional Verification Language e***

- clusterii cu dimensiune foarte mică sunt: $dimC_1=9$ și $dimC_5=13$ fața de o medie de peste 20 de elemente
- clusterii cu gradul de împrăștiere cel mai mare sunt: C_7, C_8 și C_9 - de 3000 – 7000 de unități
- clusterii care au valori de extrem ori foarte mari ori foarte mici pentru relevanță sunt: C_1 și C_2 (valori foarte mici)
- distibuția centrozilor față de media componentelor clusterilor indică o valoare de extrem pentru clusterii C_2 și C_6
- clusterii cei mai relevanți sunt C_3 și C_4
- vom alege ca și cluster cel mai relvant pe C_3, deoarece are o valoare mai bună la parametrul relevanță decât C_4

b) ***C++ Language Tutorial***

- clusterii cu dimensiune foarte mică sunt: $dimC_5=8$, $dimC_8=10$ și $dimC_4=12$ față de o medie de peste 28 de elemente
- clusterii cu gradul de împrăștiere cel mai mare sunt: C_2 și C_7 - de 1200 – 1500 de unități
- distibuția centrozilor față de media componentelor clusterilor indică o valoare de extrem pentru clusterii C_5 și C_8
- clusterul cu relevanța foarte mare în raport cu celelalte valori este pentru C_4 iar clusterii cu relevanța foarte mică sunt C_3 și C_6 (Tabel 4.15.)
- clusterul cel mai relevant devine C_1

c) ***The JavaTM Language Specification Third Edition***

- clusterii cu dimensiune foarte mică sunt: $dimC_3=2$, $dimC_4=10$ și $dimC_7=15$ fața de o medie de peste 65 de elemente
- clusterii cu gradul de împrăștiere cel mai mare sunt: C_2, C_6 și C_7 - de 5000-8000 de unități
- clusterul care are valori de extrem foarte mici pentru factorul relevanță este: C_3
- distibuția centrozilor față de media componentelor clusterilor indică o valoare de extrem pentru C_5 și C_6
- clusterul cel mai relevant devine C_1

d) ***SystemVerilog 3.1a Language Reference Manual***

- clusterii cu dimensiune foarte mică sunt: $dimC_6=4$, $dimC_1=16$ și $dimC_7=20$ fața de o medie de peste 55 de elemente
- clusterii cu gradul de împrăștiere cel mai mare sunt: C_3, C_4, C_5 și C_8 - de 3000-5000 de unități
- clusterul care are valori de extrem foarte mici pentru factorul relevanță este: C_3
- distibuția centrozilor față de media componentelor clusterilor indică o valoare de extrem pentru

C_1 și C_7
- clusterul cel mai relevant devine C_2

e) ***PHP Tutorial From beginner to master***
- clusterul cu dimensiune foarte mică este: $dimC_1=3$ față de o medie de peste 7 de elemente
- clusterul cu gradul de împrăștiere cel mai mare este: C_1 - de 170 de unități față de o medie de 30-60
- distibuția centrozilor față de media componentelor clusterilor nu indică o valoare de extrem pentru cei trei clusteri
- clusterul cu relevanța foarte mică în raport cu celelalte valori este pentru C_1 (Tabel 4.15.)
- clusterii cei mai relevanți sunt C_2 și C_3

4.2.4. Extragere de informații despre structuri sintactice

Observații:
- algoritmul propus presupune cunoașterea apiori a conceptelor din cadrul limbajului de programare
- conceptele descrise formează un lexicon pe baza căruia sunt construite șabloanele de căutare
- din punct de vedere al limbajelor de programare lexiconul conține cuvinte ce descriu structuri de date și acțiuni ce pot fi implementate de către utilizator

Algoritmul propus a fost testat pe un singur document de intrare listat mai jos, pentru o serie de structuri sintactice ce sunt descrise în continuare:
- *Draft Standard for the Functional Verification Language e* (Design Automation Standard Committee, 2007)

Conceptele definite în cadrul documentului de intrare sunt definite în Fig. 5., ordinea evidențiată în cadrul figurii reprezentând ierarhia lor în cadrul codului sursă valid din respectivul limbaj de programare.

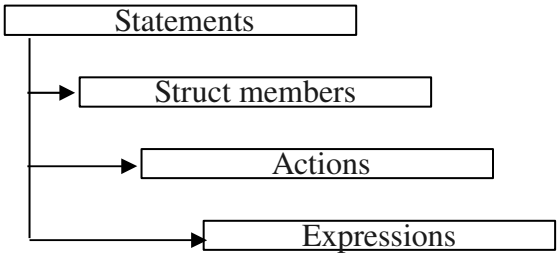

Fig. 5. Conceptele cheie ale limbajului de programare "e"

Algoritmul propus pentru extragerea sintaxei structurilor sintactice a fost aplicat pe câte un set de structuri din categoriile de mai sus. Definim câte un lexicon pentru fiecare din cele patru concepte de mai sus. Algoritmul are în vedere identificarea sintaxei corespunzătoare fiecărei structuri sintactice pe care o primește ca parametru și transpunerea acestei sintaxe într-o formă apropiată de modul în care trebuie scrisă o regulă sintactică în limbajul ANTLR.

Pentru aceasta se definesc următoarele reguli:

- cuvintele cheie care se întâlnesc în sintaxa structurilor sintactice sunt identificate și păstrate nealterate
- cuvintele care nu fac parte din categoria cuvintelor cheie sunt reduse la noțiunea de identificator
- operatorii sunt identificați și traslatați la un set comun de nume folosit în cadrul parserului ANTLR
- operatorul "|" face parte din categoria operatorilor care ar putea să nu facă parte prorpiu-zisă din sintaxă, ci să fie un marcator pentru opțiuni în cadrul sintaxei

În tabelele următoare sunt prezentate rezultatele obținute pentru diverse structuri sintactice și encodarea lor conform regulilor definite anterior.

Sintaxă structură sintactică în documentul de intrare	Sintaxă structură sintactică descrisă în limbajul propiu
struct *struct-type* [**like** *base-struct-type*] { [*struct-member*; ...]}	`'struct' ID LBRACKET 'like' ID RBRACKET LBRACE LBRACKET ID DOT ID`
extend [*struct-subtype*] *base-struct-type* { [*struct-member*; ...]}	`'extend' LBRACKET ID RBRACKET ID LBRACE LBRACKET ID DOT ID`
unit *unit-type* [**like** *base-unit-type*] { [*unit-member*; ...]}	`'unit' ID LBRACKET 'like' ID RBRACKET LBRACE LBRACKET ID DOT ID`

Tabel 8. Translatarea la limbajul propriu a structurilor sintactice din cadrul conceptului "statements"

Sintaxă structură sintactică în documentul de intrare	Sintaxă structură sintactică descrisă în limbajul propiu
Field: [**package** | **protected** | **private**] [**const**] [**!**] [**%**] *field-name*[**:** *type*] [[*min-val* .. *max-val*]] [(**bits** | **bytes**):*num*)]	`LBRACKET 'package' '|' 'protected' '|' 'private' RBRACKET LBRACKET 'const' RBRACKET LBRACKET LOGIC_NOT RBRACKET LBRACKET PERCENT RBRACKET ID LBRACKET COLON 'type' RBRACKET LBRACKET LBRACKET ID DOT DOT ID RBRACKET RBRACKET LBRACKET LPAREN LPAREN 'bits' '|' 'bytes' RPAREN COLON NUMBER RPAREN RBRACKET`
Temporal: **event** *event-type*[**is** [**only**] *temporal-expression*]	`'event' ID LBRACKET 'is' LBRACKET 'only' RBRACKET ID RBRACKET`

Subtype: **when** [*struct-subtype*] *base-struct-type* **{** [*struct-member*; ...]**}**	'when' LBRACKET ID RBRACKET ID LBRACE LBRACKET ID SEMICOLON DOT DOT DOT RBRACKET RBRACE
Coverage: **cover** *event-type* [**using** *coverage-group-option*, ...] **is** {*coverage-item-definition*; ...}	'cover' ID LBRACKET 'using' ID DOT DOT DOT RBRACKET 'is' LBRACE ID SEMICOLON DOT DOT DOT RBRACE
Method: *method-name* **(**[*parameter-list*]**)** [**:** *return-type*] **is** [**inline**] {*action;...*}	ID LPAREN (parameter_list)? RPAREN (COLON type)? "is" ("inline")? LBRACE actions RBRACE SEMICOLON
Method: *method-name* **(**[*parameter-list*]**)** [**:** *return-type*]**@***event* **is** {*action;...*}	ID LPAREN (parameter_list)? RPAREN (COLON type)? AT ID "is" LBRACE actions RBRACE SEMICOLON
Method: *method-name* **(**[*parameter-list*]**)** [**:** *return-type*] [**@***event-type*] **is** (**also**|**first**|**only**|**inline only**) {*action;...*}	ID LPAREN (parameter_list)? RPAREN (COLON type)? (AT ID)? "is" LPAREN "also" | "first" | "only" | "inline" | "only" RPAREN LBRACE actions RBRACE SEMICOLON

Tabel 9. Translatarea la limbajul propriu a structurilor sintactice din cadrul conceptului "struct members"

Sintaxă structură sintactică în documentul de intrare	**Sintaxă structură sintactică descrisă în limbajul propiu**
Variables: **var** *name* [**:** [*type*] [**=** *exp*]]	"var" ID (COLON (type)? (EQ expressions)?)? SEMICOLON
Invoke methods: **compute** [[*struct-exp*].]*method-name*([*parameter-list*])	"compute" ((expressions)? DOT)? ID LPAREN (parameter_list)? RPAREN SEMICOLON
Emit: **emit** [*struct-exp.*]*event-type*	"emit" (ID DOT)? ID SEMICOLON
Flow: **break**	'break' SEMICOLON

Flow: **continue**	`'continue' SEMICOLON`
Iterations: **while** *bool-exp* [**do**] {*action; ...*}	`"while" expressions ("do")? LBRACE actions RBRACE SEMICOLON`
Iterations: **for each** [*type*] [(*item-name*)] [**using index** (*index-name*)] **in** [**reverse**] *list-exp* [**do**] {*action*; ...}	`"for" "each" (type)? ( LPAREN ID RPAREN)? ("using" "index" LPAREN ID RPAREN)? NUMBER ("reverse")? expressions ("do")? LBRACE actions RBRACE SEMICOLON`
Iterations: **for each** [**line**] [(*name*)] **in file** *file-name-exp* [**do**] {*action*; ...}	`"for" "each" ("line")? (LPAREN ID RPAREN)? NUMBER "file" expressions ("do")? LBRACE actions RBRACE SEMICOLON`
Iterations: **for** *var-name* **from** *from-exp* [**down**] **to** *to-exp* [**step** *step-exp*] [**do**] {*action*; ...}	`"for" ID "from" expressions ("down )? "to" expressions ("step" expressions)? ("do")? LBRACE actions RBRACE SEMICOLON`

Tabel 10. Translatarea la limbajul propriu a structurilor sintactice din cadrul conceptului "actions"

4.3. Translatare la analizorul sintactic

4.3.1. Modelarea UML a aplicației de translatare la analizorul sintactic

Analizorul sintactic implică generarea unor reguli sintactice și lexicale care să corespundă cu gramatica limbajului de programare parsat. Regulile sintactice se pot grupa în două clase: reguli care pot fi folosite în cadrul oricărui analizor sintactic și reguli specifice fiecărui analizor sintactic. Regulile comune implică informații despre tratarea spațiilor, a șirurilor de litere și a identificatorilor. Regulile specifice se referă la caracterele speciale care pot să apară în limbajul de programare (operatori), comentarii. Regulile sintactice sunt specifice fiecărui limbaj de programare, urmându-i structura ierarhică.

Generatorul pentru analizorul sintactic corespunzător limbajului de programare descris în tutorialul *Draft Standard for the Functional Verification Language e* este dezvoltat în limbajul Perl. (Allen, 2011) Arborele gramaticii este generat în limbajul ANTLR. (Parr, 2007)

Actorii implicați în aplicație sunt:

- utilizatorul – solicită aplicației module, pentru a genera analizorul sintactic
- rezultatele obținute de la sistemul IE de extragere de informații despre sintaxa structurilor sintactice

Diagrama cazurilor de utilizare este descrisă de Fig. 6., fiind realizată cu ajutorul utilitarului Gaphor

(Gaphor, 2011):

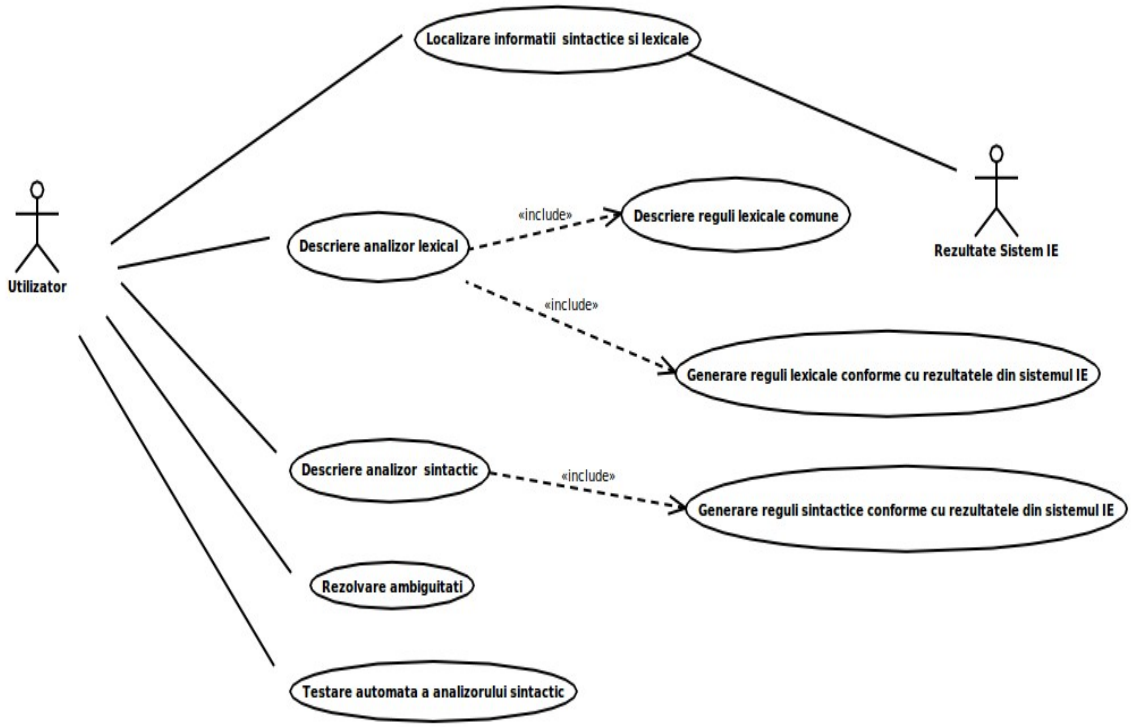

Fig. 6. Diagrama cazurilor de utilizare

4.3.1. Algoritm translatare automată la analizorul sintactic

Propunem următorul algoritm pentru translatarea regulilor sintactice și lexicale extrase din documentul de intrare la codul analizorului sintactic implementat în limbajul ANTLR.

Algoritm translatare la codul unui analizor sintactic

Pas 1.

- Generare headere

Pas 2.

- Generare reguli sintactice
  - generare opțiuni specifice
  - încărcare fișiere de date ce conțin codificarea structurilor sintactice la limbajul comun înteles de generatorul de cod
  - procesare fișiere de date conform cu regulile sintactice pe care le urmează limbajul translatat

Pas 3.

- Generare reguli lexicale
  - generare opțiuni specifice
  - generare reguli lexicale comune
  - încărcare fișiere ce conțin cuvintele cheie, operatorii și comentariile limbajului ce

urmează a fi translatat

- generare operatori limbaj împreună cu precedența acestora conform cu fișierul de date specific
- generare cuvinte cheie limbaj conform cu fișierul de date specific
- generare comentarii limbaj conform cu fișierul de date specific

Descriere detaliată a pașilor algoritmului:
a) Generare reguli sintactice

Regulile sintactice sunt generate în mod automat pe baza unor fișiere de date grupate pe categorii, fișiere obținute în urma prelucrării infomației obținute de sistemul de extragere de informații (IE) din documentul de intrare.

Fișierele de date sunt clasificate în funcție de conceptele implementate în cadrul limbajului de programare. Fiecare astfel de fișier conține pe fiecare linie câte o structură sintactică distinctă. Un exemplu (cel corespunzător structurilor sintactice din categoria *statements*) este definit mai jos:

```
'struct' ID LBRACKET 'like' ID RBRACKET LBRACE LBRACKET ID DOT ID
'extend' LBRACKET ID RBRACKET ID LBRACE LBRACKET ID DOT ID
'unit' ID LBRACKET 'like' ID RBRACKET LBRACE LBRACKET ID DOT ID
```

Regulile sintactice create sunt grupate și ele în funcție de aceleași concepte și urmează următoarele constrângeri:

- Tratează tokenii `LBRACKET/ RBRACKET` ca pe marcatori de argumente opționale
- În cazul structurilor sintactice care implică blocuri de cod interioare generează regula din gramatică până la marcatorul de început de bloc de cod interior, și anume `LBRACE` urmând să completeze regula cu numele regulii sintactice care se află în interior și să închidă la sfârșitul acesteia blocul prin tokenul `RBRACE`. De exemplu: pentru regula corespunzătoare structurilor va genera un cod asemănător celui de mai jos:

  ```
  'struct' ID LBRACKET 'like' ID RBRACKET LBRACE 'struct_members' RBRACE
  ```

- În cazul în care întâlnește structuri sintactice care conțin în declarația lor apelul altor structuri sintactice, așa cum este cazul prezentat mai sus, generează în mod automat regula folosind sintaxa `(sub_regulă)*` ceea ce înseamnă că `sub_regulă` poate să apară în cadrul codului sursă de zero sau mai multe ori permițând astfel sa putem compila fără erori un cod sursă de tipul următor:

```
struct my_struct {
      a : int;
      b: bool;
}
```

Limbajul comun definit în cadrul tezei pentru formatul fișierelor de date corespunzătoare structurilor sintactice se bazează pe următoarele reguli și observații:

- Fiecare token citit este distribuit într-un din categoriile: posibili identificatori sau operatori
- Fiecare posibil identificator este distribuit într-una din clasele: cuvinte cheie, tip de date, identificator

- Fiecare operator este trecut prin tabela operatorilor

b) Generare reguli lexicale

Regulile lexicale conțin un set bine definit care poate fi aplicat oricărui tip de limbaj. Am denumit acest set de reguli în cadrul tezei prin **regulile lexicale comune**, acestea fiind:

- reguli pentru tratarea literalilor
- reguli pentru gestionarea spațiilor libere și a caracterelor speciale de tip linie nouă
- reguli pentru tratarea numerelor
- reguli pentru tratarea identificatorilor

Cuvintele cheie ale limbajului sunt citite din fișierul de date obținut în urma aplicării algoritmului de tip IE pentru extragerea acestora, formatul fiind câte un cuvânt pe fiecare linie. Algoritmul de generare a gramaticii scrie reguli de tipul următor pentru fiecare din cuvintele întâlnite în fișierul de date:

```
LABEL
      :      'label'
      ;
```

4.3.3. Analiza ambiguităților detectate

Un limbaj este ambiguu dacă aceeași propoziție sau frază poate fi interpretată în mai multe moduri. Pentru domeniul calculatoarelor, o ambiguitate tipică este cea pentru declarația lui *if-then-else*, unde clauza *else* poate fi atașată celui mai recent bloc *if-then* ori unui bloc *if-then* de pe ramură superioară. Manualele de specificații pentru limbajele de programare rezolvă acest tip de ambiguitate prin declararea următoarei reguli: clauza *else* se închide de fiecare dată cu cel mai recent bloc *if-then*.

Non-determinism

Un analizor sintactic este non-determinist dacă există cel puțin un punct de decizie în care acesta nu poate rezolva drumul pe care trebuie să-l parcurgă. Problemele legate de non-determinism apar datorită unei slabe parsări a șirului de intrare. (Parr, Glossary)

Dacă strategia aleasă în scrierea regulilor funcționează doar pentru gramatici care nu sunt ambigue, atunci gramaticile ambigue vor conduce la parsere non-deterministe – aceasta fiind una din strategiile de bază pentru LL și LR. Chiar și gramaticile care nu sunt ambigue pot genera parsere non-deterministe.

Pentru regula de mai jos:

```
declaration
      :     ID ID SEMICOLON
      |     ID SEMICOLON
      ;
```

avem un parser de tip LL(2) deoarece cel de-al doilea simbol de lookahead (ori `ID` ori `SEMICOLON`) determină în mod unic care alternativă trebuie urmată. Se poate folosi metoda de factorizare la stânga pentru a reduce această regulă.

Factorizarea la stânga

Presupunem că avem următoarea regulă: (Luber, 2009)

```
a : L+ K
  | L+ M
  ;
```

ANTLR nu poate decide care alternativă trebuie urmată și în consecință o exclude pe cea de-a doua. Prin factorizare la stânga, regula se poate transforma astfel:

```
a : L+ (K | M)
  ;
```

Factorizarea la stănga se poate aplica chiar și între mai multe reguli. Pentru setul de reguli din stânga putem rescrie ca în codul din partea dreaptă:

```
a : b
  | c
  ;
b : L+ K
  ;
c : L+ M
  ;
```

```
a : L+ (b | c)
  ;
b : K
  ;
c : M
  ;
```

În cadrul gramaticii obținute în mod automat pornind de la interpretarea fișierelor de date rezultate în urma aplicării algoritmilor de tip extragere de informații au fost identificate ambiguități pe care compilatorul de ANTLR le-a marcat ca fiind erori de tip *non-determinism*.

Pentru regula de tip *statements* avem următoarea sintaxă:

```
statement
    :    "struct" ID ( "like" ID  )? LBRACE ( struct_members )* RBRACE
    |    "extend" ( ID  )? ID LBRACE ( struct_members )* RBRACE
    |    "unit" ID ( "like" ID  )? LBRACE ( struct_members )* RBRACE
    ;
```

Ambiguitatea detectată este pentru simbolurile marcate cu bold – aceasta putând fi rezolvată prin aplicarea factorizării la stânga, transformând regula în felul următor:

```
statement
    :    "struct" ID ( "like" ID  )? LBRACE ( struct_members )* RBRACE
    |    "extend" ( ID ) + LBRACE ( struct_members )* RBRACE
    |    "unit" ID ( "like" ID  )? LBRACE ( struct_members )* RBRACE
    ;
```

O altă modalitate de a rezolva ambiguitățile în cadrul gramaticii este folosirea de predicate sintactice.

Predicate sintactice

Proprietățile predicatelor sintactice sunt: (Colaiuta, 2007)

- validează sintaxa prin aplicarea unei alternative
- setează contextul sintactic care trebuie satisfăcut pentru ca o regulă să fie urmată

- scanează în mod automat șirul de intrare pentru mai multe simboluri pentru a ajuta la luarea de decizii (*lookahead* arbitrar)
- ordonează alternativele dintr-o regulă (specificând precedența regulilor care altfel ar fi ambigue)
- prima regulă care poate fi aplicată va fi aplicată
- indică unde poate apărea *backtracking-ul* în gramatică
- poate apărea doar în partea stângă a unei alternative
- într-o regulă cu mai multe alternative, ultima regulă nu necesită un predicat explicit
- într-o regulă cu mai multe alternative dacă unele reguli nu sunt mutual ambigue atunci nu necesită predicate
- se poate folosi opțiunea de *backtrack = true* care permite ANTLR-ului să insereze în mod automat predicate sintactice alternativelor pentru care utilizatorul nu a specificat predicatelor
- încărcarea dată de *backtracking* poate fi amortizată prin setarea opțiunii *memoize = true* de preferat nu global, ci per regulă
- acțiunile nu sunt executate pe timpul *backtracking*-ului

Pentru setul de reguli descris ca exemplu la factorizarea la stânga, aplicarea predicatelor sintactice modifică regula inițială astfel: (Luber, 2009)

```
a : (L K) => b
  | c
  ;
b : L K
  ;
c : L M
  ;
```

Ambiguitatea detectată în cadrul gramaticii pentru care se pot aplica predicate sintactice este dată de următoarea regulă:

```
struct_members
    :    ("package" | "protected" | "private")? ("const")? (LOGIC_NOT)? (PERCENT)?
ID (COLON type)? ((ID DOTDOT ID)?)? (LPAREN LPAREN "bits"|"bytes" RPAREN COLON
NUMBER RPAREN )?  SEMICOLON
    |    "event" ID ( "is" ( "only"  )? ID  )?  SEMICOLON
    |    method_declaration
    ;
method_declaration
    : ID LPAREN (parameter_list)? RPAREN (COLON  type)? "is" ("inline")? LBRACE
NUMBER RBRACE SEMICOLON
    | ID LPAREN (parameter_list)? RPAREN (COLON type)? AT ID "is" LBRACE NUMBER
RBRACE SEMICOLON
    | ID LPAREN (parameter_list)? RPAREN (COLON type)? (AT ID)? "is" LPAREN "also"
| "first" | "only" | "inline" RPAREN LBRACE NUMBER RBRACE SEMICOLON
    | ID LPAREN (parameter_list)? RPAREN (COLON type)? (AT ID)? "is" LPAREN
"undefined" | "empty" RPAREN  SEMICOLON
    ;
```

O primă problemă este legată de secvența de la regula *method_declaration* deoarece din cele patru alternative propuse nu se poate lua decizia care trebuie urmată datorită secvenței comune care este marcată prin bold-uirea textului. Ambiguitatea se poate rezolva aplicând o factorizare la stânga, regula transformându-se astfel:

```
struct_members
    :    ("package" | "protected" | "private")? ("const")? (LOGIC_NOT)? (PERCENT)?
```

```
ID (COLON type)? ((ID DOTDOT ID)?)? (LPAREN LPAREN "bits"|"bytes" RPAREN COLON
NUMBER RPAREN )?  SEMICOLON
    |    "event" ID ( "is" ( "only"  )? ID  )?  SEMICOLON
    |    method_header method_declaration
    ;
method_declaration
   : "is" ("inline")? LBRACE actions RBRACE SEMICOLON
   |  AT ID "is" LBRACE actions RBRACE SEMICOLON
   | (AT ID)? "is" LPAREN "also" | "first" | "only" | "inline" RPAREN LBRACE
actions RBRACE SEMICOLON
    ;
method_header
    : ID LPAREN (parameter_list)? RPAREN (COLON  type)?
    ;
```

Se poate observa că problema ambiguității este rezolvată parțial, rămânând încă o ambiguitate la nivelul regulii de *method_declaration* datorită faptului că nu se poate decide pe care alternativă să continue procesarea din cauza alternativei cu `(AT ID)? "is".` O posibilitate este comasarea celor trei alternative într-o singură alternativă. O altă ambiguitate majoră care se poate observa este legată de tokenul *ID* din regula *struct_members* și cel din regula *method_header*. Pentru rezolvarea acestei ambiguități am propus o soluție bazată pe predicate sintactice.

```
struct_members
    : (ID  LPAREN ) => method_declaration
    |  ("package"|"protected"|"private")?  ("const")?  (LOGIC_NOT)?  (PERCENT)?  ID
(COLON  type)?  (ID  DOTDOT  ID)?  (LPAREN  ("bits"|"bytes")  COLON  NUMBER  RPAREN)?
SEMICOLON
    |    "event" ID ("is"("only")? ID)? SEMICOLON
    ;
method_declaration
     :    ID LPAREN (parameter_list)? RPAREN (COLON type)?  (AT ID)? "is" ("also"
|"first"|"only"|"inline")? LBRACE actions RBRACE
    ;
```

4.4. Platforma de testate automată a analizorului sintactic

4.4.1. Arhitectura platformei de testare

Platforma de testare automată a analizorului sintactic conține următoarele module: (Fig. 7.)

- Modul încărcare teste în structura de makefile-uri (Yoshino, 2000)
  - pentru tipul de teste dorite se creează regula din fișierul de makefile care specifică locația și numele fișierelor de test
- Rularea testelor
- Procesarea rezultatelor testării
  - rezultatele obținute sunt stocate în fișiere text cu următorul format :
    - categorie teste
    - tip structură sintactică
    - tip test pozitiv/negativ
    - rezultate ANTLR
    - admis/respins
- Generarea de rapoarte de testate

- rapoartele de testare sunt generate sub forma de pagini HTML ce descriu rezultatele testării
- se pot selecta:
  - categoria structurii sintactice
  - numele structurii sintactice
  - direcția testelor

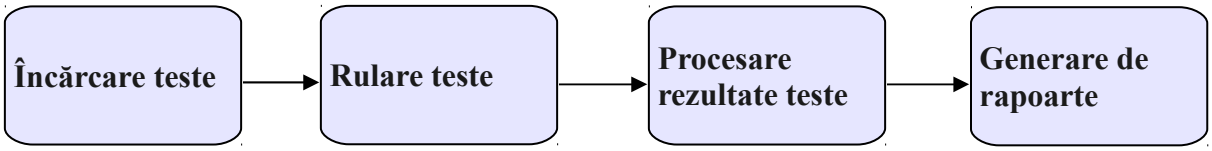

Fig. 7. Arhitectura platformei de testare automată a analizorului sintactic

4.4.2. Testarea și validarea analizorului sintactic

Validarea unui analizor sintactic presupune testarea fiecărei reguli din gramatica analizată. Pentru aceasta se propune un set de teste care să acopere toate cazurile de utilizare ale unei reguli, teste care verifică atât faptul că un cod sursă valid este acceptat de către regulile gramaticii cât și cod invalid care este rejectat de către gramatică. Pentru testatea analizorului sintactice am implementat o serie de 135 teste (Tabel 11.)

Categorie structură sintactică	**Nume structură sintactică**	**Teste**	
		Pozitive	**Negative**
Statement	`'struct'`	2	2
	`'unit'`	2	2
	`'extend'`	2	2
Struct member	`'event'`	3	7
	`'field'`	8	10
	`'method'`	10	10
Action	`'var'`	5	5
	`'compute'`	5	5
	`'emit'`	2	3
	`'while'`	5	5
	`'for'`	22	24
	`'break'`	3	4
	`'continue'`	3	4

Tabel 11. Validarea analizorului sintactic

Testele pozitive/valide au în vedere următoarele aspecte:

- folosirea tuturor opțiunilor pe care le permite structura sintactică în diverse combinații funcție de numărul secvențelor opționale

- combinarea de diverse structuri sintactice în vederea elaborării unui cod sursă mai complicat și mai apropiat de nevoile utilizatorului

Testele negative/invalide au în vedere următoarele aspecte:
- introducerea de opțiuni inexistente
- apelarea mai multor opțiuni, dintre care doar anumite secvențe reprezintă cod valid, în combinații nepermise
- declararea de tipuri de structuri sintactice în contexte sintactice care nu le suportă
- omiterea de paranteze rotunde la declarația funcțiilor
- omiterea acoladelor la marcarea blocurilor de instrucțiuni
- folosirea de cuvinte cheie în declarații ce permit doar tipul identificator
- utilizarea tipurilor de date în declarații care nu returnează un tip de date sau nu pot fi definite ca aparținând unui tip de date
- folosirea de comentarii linie care nu sunt suportate de limbajul de programare analizat

5. CONCLUZII ȘI CONTRIBUȚII

Lumea în care evoluăm presupune înțelegerea și acumularea unei cantități imense de informație care apoi necesită procesare. Informația este împărțită în diferite surse, necesitând un modul de integrarea a acestei informații și de sinteză a ei. A apărut astfel necesitatea unor aplicații inteligente, capabile să proceseze în mod automat informația primită sau să colecteze în mod automat date despre un subiect dat. Aceste aplicații folosesc algoritmi de clusterizare care reprezintă unul din cele mai utile instrumente din procesul de *Data Mining* pentru a descoperi grupuri și a identifica șabloane în cantități mari de date de intrare. Totodată, datorită experienței obținute de-a lungul timpului în domeniul aplicațiilor software tendința care se impune din ce în ce mai mult este de automatizare a proceselor care se pretează la acest lucru, economisind astfel timp prețios al dezvoltatorilor umani, timp care poate fi apoi folosit în crearea de noi concepte, în proiectarea de noi arhitecturi.

În contextul descris propunem studiul prezentat în cadrul tezei numit generare automată de cod sursă pentru aplicații care descriu analizoare sintactice. Scopul principal al tezei este minimizarea efortului depus de dezvoltatorii umani de aplicații din domeniul compilatoarelor prin generarea automată a secțiunilor corespunzătoare analizei sintactice și analizei lexicale din partea de *Front-End* a unui compilator pornind de la un document de intrare semi-structurat scris în limbaj natural care reprezintă un manual de referință pentru limbajul de programare în cauză.

Pentru atingerea obiectivelor tezei am proiectat și implementat o platformă de analiză și generarea formată din două subsisteme. Primul din cele două subsisteme este responsabil cu colectarea informațiilor din documentele de intrare, informații care vizează sintaxele structurilor sintactice și lexicale ale limbajului de programare și cu translatarea lor la un limbaj comun, unificat. Cel de-al doilea subsistem este responsabil cu generarea codului necesar componentelor de analiză lexicală și analiză sintactică prin translatarea informațiilor din limbajul comun obținut la etapa de extragere de informații în reguli ale gramaticii pentru analizorul sintactic țintă.

Rezultatele experimentale au avut în vedere două mari direcții: validarea informațiilor extrase cu sistemul de extragere de informații propus și validarea și testarea codului generat automat pentru

analizorul sintactic proiectat. Pentru direcția de testare și validare a sistemului de extragere de informații am propus o serie de cinci documente de intrare ce reprezintă manuale de referință pentru cinci limbaje de programare diferite; pentru validarea datelor obținute am aplicat algoritmi de clusterizare pentru rezultatele cărora am propus o serie de parametri ce reprezintă indici de calitate. Pentru direcția de testare și validare a analizorului sintactic generat automat am propus o serie de 135 de teste grupate într-o platformă de teste ce cuprinde teste pozitive (teste ce reprezintă cod sursă valid în cadrul limbajului de programare și trebuie să fie acceptat de gramatica limbajului) și teste negative (teste ce reprezintă cod sursă invalid în cadrul limbajului de programare și trebuie să fie rejectat de gramatica limbajului). În urma testelor efectuate am ajuns la concluzia că acuratețea algoritmilor de extragere de informație este ridicată, iar precizia algoritmului de generare automată a analizorului este maximă.

Principala contribuție adusă de teză este dezvoltarea de algoritmi de tip extragere de informație aplicați pe documente de intrare semi-structurate, vizând informații bine definite din cadrul acestora referitoare la sintaxa unor structuri de date și structuri sintactice ale limbajelor de programare descrise în respectivele documente. O a două contribuție este generarea automată a analizorului sintactic pentru limbajul de programare descris în documentul de intrare pornind de la rezultatele extrase din acesta.

O prezentare generală a pricipalelor contribuții aduse în cadrul tezei este:

1. Analiza și sinteza principalelor metode folosite pentru procesarea de text scris în limbaj natural și clusterizarea datelor
    1. sinteza metodelor folosite pentru extragerea de informații
    2. sinteza pentru analiza web semantică
    3. sinteza metodelor folosite pentru regăsirea de informații
    4. sinteză asupra metodelor de clusterizare cu accent pe câțiva algoritmi de clusterizare, dintre care cel de partiționare este folosit în cadrul rezultatelor experimentale
    5. sinteză a metricilor de tip distanță aplicabile rezultatelor clusterizării
    6. sinteză a studiilor de caz ce folosesc metode de tip extragere și regăsire de informații

2. Analiza și sinteza principalelor concepte utilizate în cadrul proiectării analizoarelor sintactice
    1. sinteză asupra analizorului lexical, definind rolul și componentele sale
    2. sinteză asupra analizorului sintactic, definind rolul său
    3. sinteză asupra tipurilor de ambiguități ce pot fi întâlnite în cadrul unei gramatici
    4. sinteză a principalelor soluții de eliminare a ambiguităților din cadrul unei gramatici
    5. sinteză asupra tipurilor de analizoare sintactice
    6. sinteză asupra instrumentelor software ce pot fi folosite pentru a proiecta un analizor sintactic

3. Proiectarea unui sistem IE (Information Extraction) care conține un set de algoritmi dedicați extragerii de informații lexicale și sintactice din documentele de intrare
    1. definirea unui set de parametrii pe baza cărora se ia decizia pentru cel mai relevant rezultat al algoritmilor de extragere de informație
    2. definirea unei distanțe pentru cuvintele cheie ale unui limbaj de programare, distanță ce reflectă relevanța unui cluster în raport cu ceilalți clusteri obținuți
    3. definirea unei distanțe pentru operatorii unui limbaj de programare, distanță ce reflectă relevanța unui cluster în raport cu ceilalți clusteri obținuți

4. definirea unui parametru de tip indice de calitate al agoritmului

4. Proiectarea unui generator de cod care realizează translatarea la un analizor sintactic pornind de la documente de intrare ce descriu tutoriale ale unor limbaje de programare
   1. definirea unui limbaj comun pentru descrierea rezultatelor obținute cu ajutorul algoritmilor din sistemul IE proiectat, ușor de scriptat
   2. modelarea UML a principalelor componente ale aplicației: diagrama cazurilor de utilizare și diagrama pachetelor
   3. proiectarea unui algoritm care citește fișiere de date în formatul limbajului comun propus și generează automat gramatica pentru limbajul de programare descris în documentul de intrare
   4. analiză a ambiguităților detectate de compilatorul de ANTLR pentru gramatica obținută
   5. implementarea de soluții în vederea eliminării ambiguităților detectate

5. Proiectarea unei platforme de testate automată a gramaticii rezultate
   1. proiectarea și implementarea unei arhitecturi pentru platforma de testare
   2. proiectarea de seturi de teste pozitive și negative corespunzătoare fiecărei componente generate
   3. proiectarea și dezvoltarea unei platforme Web pentru generarea de rapoarte configurabile pentru afișarea rezultatelor testării și validării analizorului sintactic generat automat

Ca și dezvoltări ulterioare propunem:
- Extinderea analizei rezultatelor clusterizarii pentru informațiile identificate de sistemul IE la noi algoritmi de clusterizare și noi metrici de distanțe
- Extinderea numărului de parametri care definesc factorii de calitate propuși pentru determinarea clusterului cel mai relevant
- Extinderea structurilor sintactice pentru care se aplică algoritmul de generare reguli sintactice și crearea de noi teste care să acopere noile reguli propuse
- Rezolvarea automată a ambiguităților detectate de ANTLR la compilarea fișierului ce conține gramatica limbajului

Bibliografie

[1] (Accellera Organization, 2004) Accellera Organization – *SystemVerilog 3.1a Language Reference Manual* – Accellera's Extensions to Verilog www.eda.org/sv/SystemVerilog_3.1a.pdf

[2] (Aho et al., 1986) Aho A., Sethi R., Ullman J. - *Compilers Principles, Techniques, and Tools*, Addison-Wesley Publishing Company, 1986

[3] (Aho et al., 1989) Aho A.V., Ganapathi M., Tjiang S.W.K. - *Code generation using tree matching and dynamic programming* – ACM Transactions on Programming Languages and Systems 11, 4, pp. 491-516, 1989

[4] (Allen, 2011) Allen J. - *Language Reference Perl 5 version 14.1 documentation*, http://perldoc.perl.org/index-language.html

[5] (Baeza-Yates et al., 1999) Baeza-Yates R, Ribeiro-Neto B. – *Modern Information Retrieval*; ACM Press, NewYork, 1999

[6] (Böhm, 2007) Böhm I. - *Automatic Code Generation using Dynamic Programming Techniques* – Master Work Paper at Johannes Kepler Universität Linz, 2007

[7] (Califf et al., 1998) Califf M., Mooney R. – *Relational learning of pattern-match rules for information extraction* – Proceedings of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing Stanford, California, 1998

[8] (Colaiuta, 2007) Colaiuta W. - *ANTLT Predicates* – 2007 https://wincent.com/wiki/ANTLR_predicates

[9] (Design Automation Standard Committee, 2007) Design Automation Standard Committee of the IEEE Computer Society - *IEEE P1647™/D9 Draft Standard for the Functional Verification Language e* www.ieee1647.org/downloads/P1647_Draft_6_071214.pdf

[10] (Dubes et al., 1988) Dubes R.C., Jain A.K. - *Algorithms for Clustering Data*, Prentice Hall, 1988

[11] (Dubes et al., 1979) Dubes R.C., Jain A.K. - *Validity Studies in Clustering Methodologies*, Pattern Recognition, 11, pp. 235-254, 1979

[12] (Fegaras, 2005) Fegaras L. - *Design and Construction of Compilers* – CSE 5317/4305, University of Texas at Arlington, CSE http://lambda.uta.edu/cse5317/notes/notes.html

[13] (Freitag, 1998) Freitag D. – *Information extraction from HTML: Application of a general learning approach* – Proceedings of the 15th International Conference on Artificial Intelligence (IJCAI), 1998

[14] (Ganapathi, 1980) Ganapathi M. - *Code Generation and Optimization using Attribute Grammars* – PhD thesis, University of Wisconsin, Madison, 1980

[14] (Ganapathi et al., 1982) Ganapathi M., Fischer C.N. - *Description-driven code generation using attibute grammars* – In POPL'82: Proceedings of the 9th ACM SIGPLAN-SIGACT symposium on Principles of programming languages, ACM Press, pp. 108-119, New York, USA, 1982

[15] (Gaphor, 2011) Gaphor Team – *Gaphor, the essence of UML modelling* - 2009-2011

[16] (Glanville, 1977) Glanville R.S. - *A machine independent algorithm for code generation and its use in retargetable compilers* – PhD Thesis, University of California, Berkeley, 1977

[17] (Glanville et al., 1978) Glanville R.S., Graham S.L. - *A new method for compiler code generation* – In POPL'78: Proceedings of the 5th ACM SIGACT-SIGPLAN symposium on Principles of programming languages, ACM Press, pp. 231-254, New York, 1978

[18] (Gosling et al., 2005) Gosling J., Joy B., Steele G., Bracha G. - *The Java™ Language Specification Third Edition* - ISSN 0-321-24678-0, Santa Clara, California, 2005

http://www.psc.informatik.uni-jena.de/languages/Java/javaspec-3.pdf

[19] (Hand et al., 2001) Hand D., Heikki M., Padhraic S. - *Principles of Data Mining* – MIT Press, 2001

[20] (Hartigan, 1975) Hartigan J.A. - *Clustering Algorithms*, Wiley, New York, 1975

[21] (Hobbs, 1994) Hobbs J. - *The Generic Information Extraction System* – Proceedings of the 5th Message Understanding Conference (MUC-5), Morgan Kaufmann Publishers, San Mateo, California, 1994

[22] (Hsu, 1998) Hsu C.N., Dung M. – *Generating finite-state transducers for semi-structured data extraction from the web* – Journal of Information Systems 23(8):521-538, 1998

[23] (Huffman, 1996) Huffman S. – *Learning information extraction patterns from examples. Connectionist, statistical, and symbolic Approaches to Learning for Natural Language Processing* – Spinger-Verlag, 1996

[24] (Kaufman et al., 1990) Kaufman L., Rousseeuw P.J. - *Finding Groups of Data*, Wiley, New York, 1990

[25] (Kim et al., 1995) Kim J., Moldovan D. – *Acquisition of linguistic patterns for knowledge-based information extraction* – IEEE Transactions on Knowledge and Data Engineering 7(5):713-724, 1995

[26] (Krupka, 1995) Krupka G. – *Description of the SRA system as used for MUC6* – Proceedings of the 6th Message Understanding Conference (MUC6), pp. 221-235, 1995

[27] (Kushmerick et al., 1997) Kushmerick N., Weld D., Doorenbos R. - *Wrapper induction for information extraction* - Proceedings of the 15th International Conference on Artificial Intelligence (IJCAI), pp. 729-735, 1997

[28] (Luber , 2009) Luber J. - *How to remove global backtracking from your grammar* – 2009

[29] (MacQueen, 1967) MacQueen J.B. - *Some Methods for Classification and Analysis of Multivariate Observations*, Proceedings of the 5th Berkely Symposium on Mathematical Statistics and Probability, Berekely, University of California Press, pp. 281-297, 1967

[30] (Marmanis et al., 2009) Marmanis H., Babenko D. - *Algorithms of the Intelligent Web* – Manning Publications , 2009

[31] (Murty et al., 1999) Murty M., Jain A., Flyn P. - *Data clustering: A review* – ACM Computing Surveys, 31(3), 1999

[32] (Muslea et al., 1999) Muslea I., Minton S., Knoblock C. – *A hierarchical approach to wrapper induction* – Proceedings of the 3rd International Conference on Autonomous Agents (AA-99), 1999

[33] (Neville, 1999) Neville P.G. - *Decision Trees for Predictive Modelling* – SAS Institute INC., 1999

[34] (Parr, 2007) Parr T. - *The Definitive ANTLR Reference: Building Domain Specific Language*, ISBN: 978-0-9787-3925-6, 2007 http://www.antlr.org/

[35] (Parr, Glossary) Parr. T – *ANTLR-centric Language Glossary* http://www.antlr.org/doc/glossary.html

[36] (PHP Community) PHP Community - *PHP Tutorial From beginner to master* - http://www.mobilitystudies.com/mob/images/stories/PHP_Tutorial_From_beginner_to_master.pdf

[37] (Proebsting, 1995) Proebsting T. - *Optimizing an ANSI C Interpretor with Superoperators* – Proceedings of Principles of Programming Languages POPL'95, pp. 280-287, San Francisco, California, 1995

[38] (Raskuti et al., 1999) Raskuti B., Leckie C. - *An Evaluation of Criteria for Measuring the Quality of Clusters*, Proceedings of the 16th International Joint Conference on Artificial Intelligence, pp. 905-910, ISBN: 1-55860-613-0, 1999

[39] (Riloff, 1993) Riloff E. – *Automatically constructing a dictionary for information extraction tasks* – Proceedings of the 11th National Conference of Artificial Intelligence (AAAI-93), 00. 811-816, AAAI Press/ The MIT Press, 1993

[40] (Slonnenger et al., 1995) Slonnenger K., Kurtz B. - *Formal Syntax and Semantics of Programming Languages*, Addison-Wesley Publishing Company, ISBN 0-201-65697-3, 1995

[41] (Soderland, 1999) Soderland S. – *Learning information extraction rules for semi-structured and free text* - Journal of Machine Learning, 34(1-3):233-272, 1999

[42] (Soulie, 2007) Soulie J. - *C++ Language Tutorial* - http://www.cplusplus.com/doc/tutorial/ , 2007

[43] (Tan et al., 2005) Tan P.N., Steinbach M., Kumar V. - *Introduction to Data Mining* – ISBN: 0321321367, Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 2005

[44] (Ye, 2007) Ye J. - *Numerical Linear Algebra for Data Exploration – Clustering* – CSE 494 CSE/CBS 598, 2007

[45] (Yohannes et al., 1999) Yohannes Y., Hoddinott J, - *Classification and Regression Trees: An Introduction* – International Food Policy Research Institute 2033 K Street, N.W., Washington, D.C., 20006, USA, 1991

[46] (Yoshino, 2000) Yoshino B. - *Make – a tutorial*, 2000 http://www.eng.hawaii.edu/Tutor/Make/

[47] (Wolfram Mathematica Documentation Center, 2011) Wolfram Mathematica Documentation Center –*Distance and Similarity Measures* – 2011 http://reference.wolfram.com/mathematica/guide/DistanceAndSimilarityMeasures.html

Publicații

[1] Năsui D.V., **Rancea I.**, Sgârciu V. – S*afe Driving Using Wireless Monitoring Units (2010) 0231-0233* - Annals of DAAAM for 2010 & Proceedings of the 21th International DAAAM Symposium, ISBN 978-3-901509-73-5, ISSN 1726-9679, pp. 231-232, Editor B. Katalinic, Published by DAAAM International Vienna Austria, Location: Vienna, Austria, 2010

[2] Năsui D.V., **Rancea I.**, Sgârciu V. - *Intelligent System for Bus School Safe Driving using Wireless Monitoring Units* - The 18th Telecommunications Forum TELFOR 2010, ISBN 978-86-7466-392-9, vol. 18, pp. 566-569, Location: Serbia, Belgrade, Novermber 23-25, 2010

[3] **Rancea I.**, Sgârciu V., Stamatescu G. – *Remote Acquisition: Processing and Generation of Digital Signals for a Reversible Counter through LabView Environment* - Annals of DAAAM for 2009 & Proceedings of the 20th International DAAAM Symposium, ISSN 1726-9679, ISBN 978-3-901509-70-4, vol. 20, pp 857-858, Editor B. Katalinic, Published by DAAAM International Vienna Austria, Location: Vienna, Austria, 2009

[4] **Rancea I.**, Sgârciu V., Dichiu D. - *A Pacient Smart Card Sollution Used for an Experimental Model in Health Care Environment* - Annals of DAAAM for 2009 & Proceedings of the 20th International DAAAM Symposium, ISSN 1726-9679, ISBN 978-3-901509-70-4, vol. 20, pp. 315-316, Editor B. Katalinic, Published by DAAAM International Vienna Austria, Location: Vienna, Austria, 2009

[5] Năsui D., Coșoi C., Sgârciu V., **Rancea I.** – *Using Wireless Monitoring for Market Research Interviewers* - Annals of DAAAM for 2009 & Proceedings of the 20th International DAAAM Symposium, ISSN 1726-9679, ISBN 978-3-901509-70-4, vol. 20, pp. 1153-1154, Editor B. Katalinic, Published by DAAAM International Vienna Austria, Location: Vienna, Austria, 2009

[6] Năsui D., **Rancea I.**, Sgârciu V., Oprea B., Tănase C., Ene C., Negulescu C. - W*ireless Monitoring of a Computerized City using SafeMobile Units, ICSNC 2009* – The Fourth International Conference on Systems and Networks Communications, IEEE Computer Society Order Number E3775, ISBN-13: 978-0-7695-3775-7, pp. 261-264, Location: Porto, Portugal, 2009

[7] **Rancea I.**, Sgârciu V., Dichiu D. - *A Case Study about Information Security Management Systems* - CSCS17 – the 17th International Conference on Control Systems and Computer Science, ISBN 2066-4451, vol. 1 pp. 381-385, Location: Bucharest, Romania, 2009

[8] Năsui D., **Rancea I.**, Sgârciu V., Dichiu D., Oprea B., Saru D., Catana I., Ceapâru M., Tănase C., Ene C., Negulescu C. - *Supervising Semi-autonomous Mobile Robots Using SafeMobile Wireless Units,* - RAAD 2009 – the 18th International Workshop on Robotics in Alpe-Adria-Danube Region, ISBN 978-606-521-315-9, Location: Brasov, Romania, 2009

[9] **Rancea I.**, Sgârciu V., Dichiu D. - *Remote Monitoring and Data Acquisition of Industrial Process Parameters through Internet* - Annals of DAAAM for 2008 & Proceedings of the 19th International DAAAM Symposium, ISBN 978-3-901509-68-1, ISSN 1726-9679, vol. 19 pp 1169-1170, Editor B. Katalinic, Published by DAAAM International Vienna Austria, Location:

Trnava, Slovakia, 2008

[10] **Rancea I.**, Sgârciu V., Stamatescu G. - *Remote Monitoring of Parameter for a Pressure Transducer through HART Protocol and LabView Environment,* Annals of DAAAM for 2008 & Proceedings of the 19th International DAAAM Symposium, ISBN 978-3-901509-68-1, ISSN 1726-9679, vol. 19 pp 1171-1172, Editor B. Katalinic, Published by DAAAM International, Vienna, Austria, Location: Trnava, Slovakia, 2008

[11] **Rancea I.**, Sgârciu V. - *Functional Verification of Digital Circuits using a Software System - A*utomation, Quality and Testing, Robotics.2008. AQTR 2008. IEEE International Conference, vol. 1, pp. 152-157, 22-25, Digital Object Identifier 10.1109/AQTR.2008.4588725, Location: Cluj-Napoca, Romania, 2008

[12] **Rancea I.**, Sgârciu V**.** - *Principles of Functional Verification for Digital Circuits* - Annals of DAAAM for 2007 & Proceedings of the 18th International DAAAM Symposium, ISBN 3-901509-58-5, ISSN 1726-9679, vol. 18, pp 637-638, Editor B. Katalinic, Published by DAAAM International, Vienna, Austria, Location: Zadar, Croatia, 2007

[13] Vlad M., Sgârciu V., **Rancea I.** - *Acquisition and Monitoring of Process Parameters Using Internet* - CSCS-15 International Conference of Control Systems and Computer Science*,* Bucharest, ISBN 973-8449-89-8, vol. 1, May 2005, Location: Bucharest, Romania, 2005