Modelare orientată obiect - de la limbă la limbaj

Coordonator:

Conf. Dr. Sabin - Corneliu Buraga

Absolvent:

Gordîn Ștefan

Cuprins

Cerințe

O aplicație care

Primește un **text scris într-un limbaj natural controlat** și un limbaj de programare.

Extrage un **model**, reprezentând concepte din paradigma orientată obiect.

Întoarce **codul** ce este asociat cu modelul extras, în limbajul de programare specificat

Cerințe

O aplicație care

- Să fie extensibilă relativ la **conceptele** pe care le caută în textul controlat
- Să fie extensibilă relativ la **limbajul de programare** în care generează cod

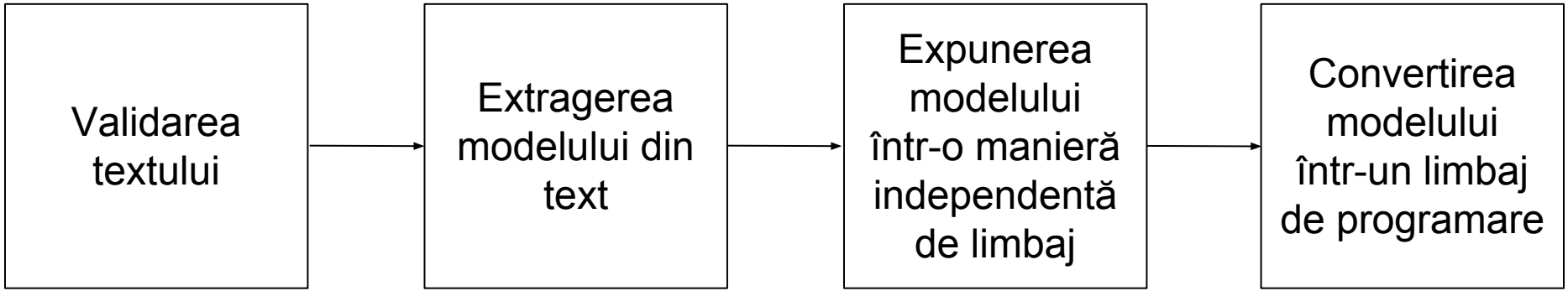
Direcții asemănătoare

Fluent editor: O aplicație care poate modela ontologii dintr-un limbaj natural controlat.

Ontologie - o listă de termeni și relații între acești termeni, definind cunoștințele dintr-un anumit domeniu, în mod structurat.

Blockly: O aplicație ce generează cod într-un limbaj de programare după o structură de tip puzzle.

Fluxul aplicației



Soluția propusă

O soluție bazată pe trei servicii web. Acestea sunt:

Serviciul de procesare de text
Primește un text scris în limbaj natural controlat, îl pre-procesează și îl validează folosind o gramatică și creează o structură numită modelul-dicționar

Serviciul de generare de cod
Primește modelul-dicționar și generează codul asociat conceptelor POO

Serviciul de integrare
Coordonează interacțiunea dintre cele două servicii descrise mai sus

Serviciul de procesare de text

Primește textul din care se va extrage modelul.

Fiecare propoziție este validată de o **gramatică** independentă de context.

```
sentence  -> descriptive_sentence
           |  possessive_sentence
           |  behavioral_sentence
           |  singleton_sentence
```

Serviciul de procesare de text

Această gramatică se bazează pe identificarea de cuvinte cheie. Spre exemplu:

- Din sintagma "C **is a** P" putem deduce relația de **moștenire**.
- Din sintagma "C **has a** V" putem deduce relația de **asociere**.
- Din sintagma "C **can** M" putem deduce un **comportament**.

Cuvintele cheie sunt folosite în cadrul gramaticii cu forma lor din dicționar. Acest fapt face posibilă interpretarea textului în orice formă sintactică.

De aceea, în faza de preprocesare, fiecare cuvânt al textului trebuie adus la forma sa din dicționar *(eng. Lemmatization)*. Exemplu:

"Bob has a car" -> "Bob have a car"
"Steve is a human" -> "Steve be a human"

Serviciul de procesare de text

În exemplul anterior, C,P,V și M joacă rolul de entități modelate. În cadrul gramaticii acestea sunt înlocuite cu părțile lor de vorbire pentru a garanta controlul asupra sensului semantic al textului. Spre exemplu:

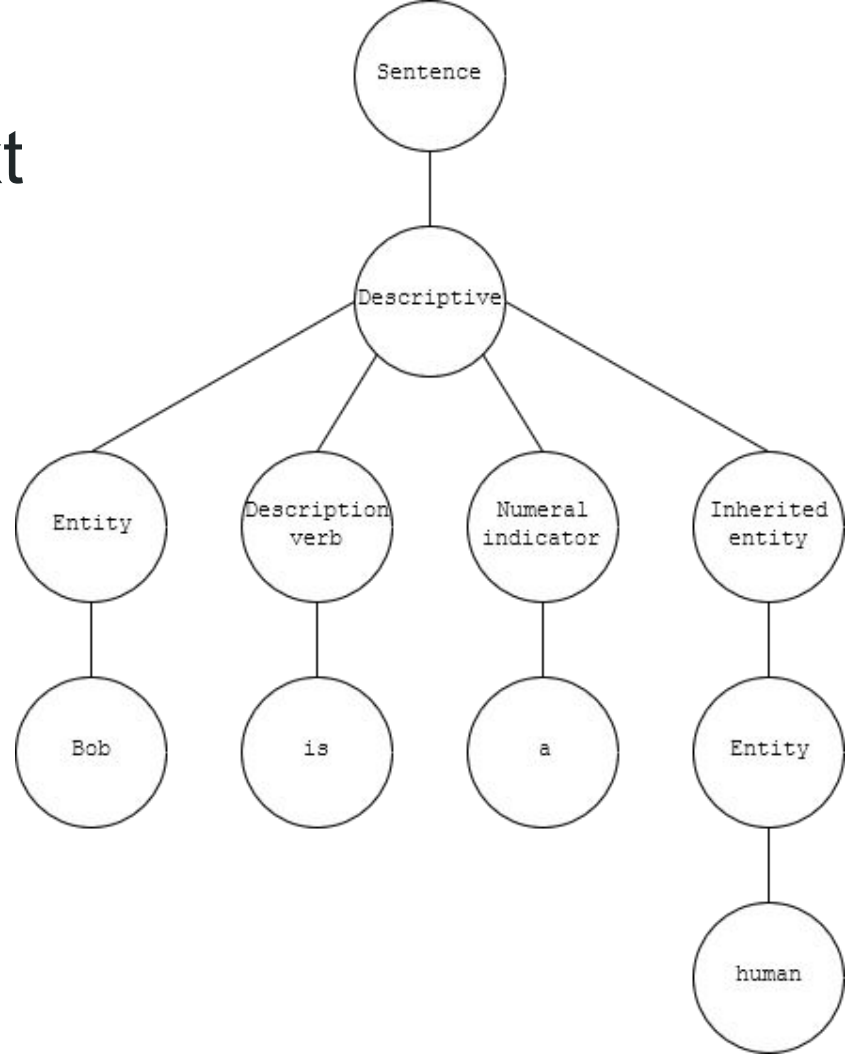
"Bob has a car" va fi înlocuită cu "substantiv **has a** substantiv" și va fi acceptată și prelucrată de către gramatică.

"Bob can beautiful" va fi înlocuită cu "substantiv can adjectiv" și va fi respinsă de către gramatică.

Serviciul de procesare de text

Cu ajutorul gramaticii, texul este controlat și transformat într-un arbore propozițional.

"Bob is a human."



Serviciul de procesare de text

Arborele este parcurs și din acesta este extras "**modelul-dicționar**".

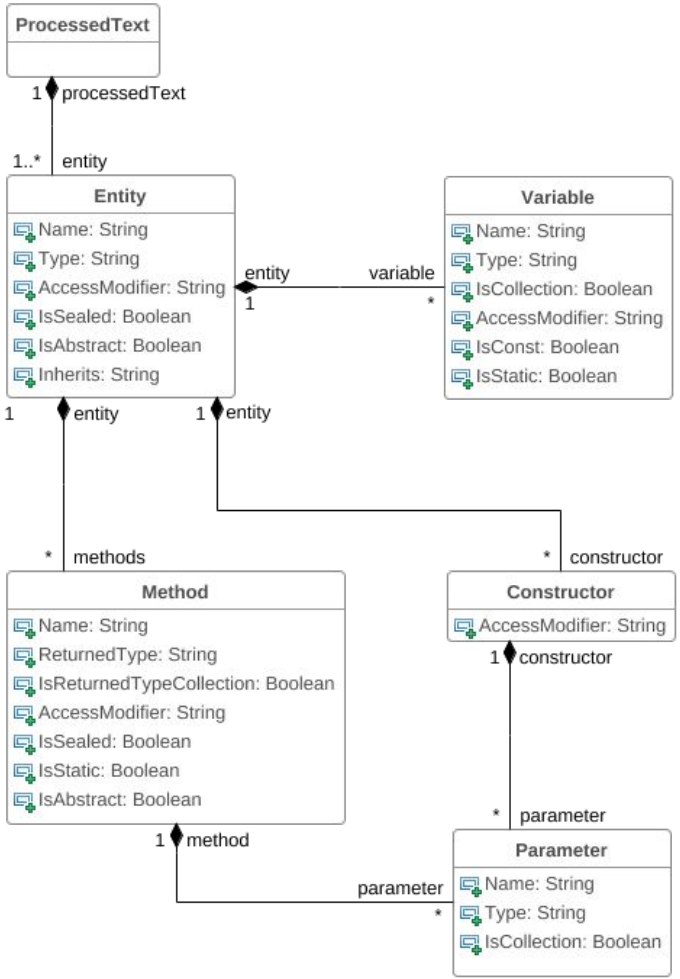
```
"Bob has a car.
 Bob's car is a Toyota.
 Bob can drive."
```

```
[{
    "name": "Bob",
    "methods": [{
        "accessModifier": "public",
        "name": "drive"
    }],
    "variables": [{
        "isCollection": false,
        "name": "car",
        "type": "Toyota",
    }]
},
{
    "name": "Toyota"
}]
```

Modelul-dicționar

Am propus modelul-dicționar pentru a reține (expresiv) un model POO, independent de un limbaj de programare.

Acesta este bazat pe structuri de tip dicționar și structuri de tip tablou, făcându-l ușor de procesat.



Serviciul de generare de cod

Primește un limbaj de programare și o structură de tip model-dicționar. Modelul-dicționar este validat sintactic și semantic pentru a nu a avea erori la compilare. Un exemplu de eroare la compilare, prevenită de **validarea sintactică**:

```
public class String {

}
```

Un exemplu de eroare la compilare, prevenită de **validarea semantică**:

```
public class Example extends Example {

}
```

Serviciul de generare de cod

Serviciul primește la intrare o structură de date independentă de vreun limbaj de programare. Astfel putem implementa independent strategii de generare pentru multiple limbaje de programare.

În cadrul acestei soluții am implementat strategii pentru limbajele Java și C#. Am tratat diferențe precum conceptul de clasă finală (în Java cuvântul-cheie "final", în C# cuvântul-cheie "sealed").

Serviciul de generare de cod

Dacă structura primită este validă, se generează codul asociat cu modelul primit în limbajul de programare specificat. Spre exemplu, dacă limbajul solicitat este **`Java`**:

`"Bob has a car. Bob's car is a Toyota. Bob can drive."`

```
public class Bob {
        public Toyota car;
        public void drive() {
                throw new UnsupportedOperationException();
        }
}
```

Serviciul de integrare

Acest serviciu concatenează fluxurile celor două servicii. Serviciului îi oferim la intrare un text și un limbaj de programare și primim clasele corespunzătoare modelului.

În plus, acesta oferă un URL către o arhivă ZIP în care se găsesc clasele generate.

Demo

Direcții de viitor

Generarea de diagrame

Implementarea de strategii pentru mai multe limbaje de programare

Extinderea gramaticii pentru acceptarea mai multor limbaje naturale (de exemplu: limba română)

Crearea fluxului invers (de la cod la limbaj natural)

Utilizarea unei arhitecturi bazată pe micro-servicii

Concluzii

Soluția propusă extrage un model bazat pe paradigma de programare orientată obiect dintr-un text natural controlat și generează cod într-un limbaj de programare.

Principalele inovații sunt:

- Crearea unei gramatici pentru extragerea conceptelor POO dintr-un text
- Crearea unei structuri de date ce reține modele într-o manieră independentă de limbaj
- Crearea unei modalități de generare de cod indiferent de limbajul de programare.

Vă mulțumesc!