**DOKUMENTACIJA**

Programski prevodioci - predmetni zadatak

**Osnovni podaci**

|  |  |  |
| --- | --- | --- |
| Broj indeksa | Ime i prezime | Tema |
| SW2/2019 | Nataša Laković | DSL za definisanje UML dijagrama u textX |

**Korišćeni alati**

|  |  |
| --- | --- |
| Naziv | Verzija |
| Python | 3.8.3. |
| textX biblioteka u python-u | 3.0.0 |

## Evidencija implementiranog dela

## U projektu su implementirane sintaksa i semantika za DSL za definisanje UML class dijagrama. U fajlu *uml\_class\_diagram.tx*definisana je gramatika jezika, a u fajlovima *custom\_classes.py* i *semantic\_check.py* implementirana je semantička provera isparsiranog koda.

## Detalji implementacije

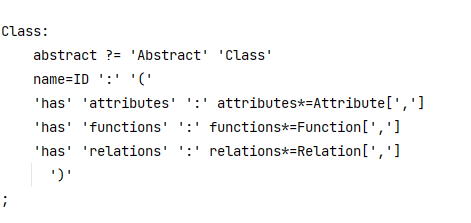
***Gramatika jezika***

Gramatika jezika je implementirana u fajlu *uml\_class\_diagram.tx.* Ideja projekta je da se omogući formulisanje class dijagrama što sličnije prirodnom jeziku i govoru ljudi pa se u skladu sa tom idejom formirala gramatika jezika.

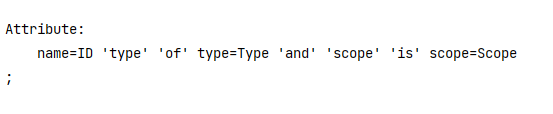
Ovaj DSL jezk se sastoji od liste elementa koji mogu biti interfejsi, enumeracije ili klase. Neophodno je da postoji barem jedan element. Svaki od elemenata ima svoje atribute, funkcije i veze ka drugim elementima, ali oni nisu obavezni za navođenje. Dakle, dozvoljeno je da neka klasa ima samo atribute ili samo funkcije itd.

Radi lakše implementacije semantičke provere, za većinu pravila postoje posebno definisane klase koje će ih prepoznavati. Klase koje su posebno implementirane su: *Interface, Enum, Class, Attribute, Function, Param, Dictionary, List, RelationWithCardinality, RelationWithoutCardinality, Cardinality, BasicType*.

Na slici 1 prikazano je pravilo za definisanje klase. Na slici se može uočiti da su atributi predstavljeni kao lista pojedinačnih atributa, isto važi i za funkcije i veze. Takođe, omogućeno je definisanje apstraktne klase koja će se od obične klase razlikovati samo po dodatom prefiku “Abstract” koji će se mapirati na boolean vrednost u instanci klase koja se formira za odgovarajuću prepoznatu klasu. Na slici 2. može se videti pravilo za navođenje atributa u sklopu interfejsa ili dijagrama klasa. Sa slike se vidi da navođenje atributa dosta liči prirodnom jeziku jer jedan primer koji bi bio u skladu sa gramatikom jezika je *“firstName type of string and scope is private”*.

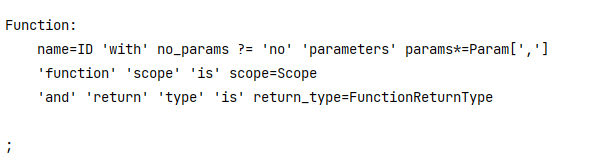


*Slika 1. Pravilo gramatike za definisanje klase*

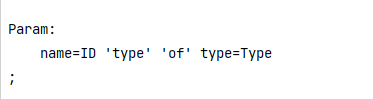


*Slika 2. Pravilo za definisanje atributa*

Kada je u pitanju definisanje funkcija, takođe se težilo da gramatika odgovara prirodnom jeziku uz nekoliko interpunkcijskih pravila. Na slici 3 možemo videti pravilo za definisanje funkcija. Omogućeno je navođenje reči ‘no’ koja naznačava da funkcija nema parametere. Takođe, na toj slici možemo primetiti da se u sklopu funkcije navode njeni parametri. Pravilo za njihovo definisanje je slično onom za definisanje atributa osim što nema opseg važenja. Slika 4 prikazuje pravilo za definisanje parametara. Primer ispravno definisane funkcije je: *setName with parameters name type of string function scope is public and return type is void*

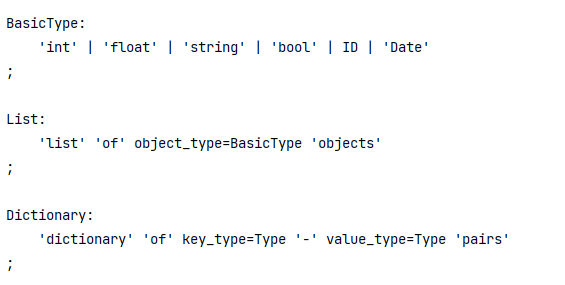


*Slika 3. Pravilo za definisanje funkcija*



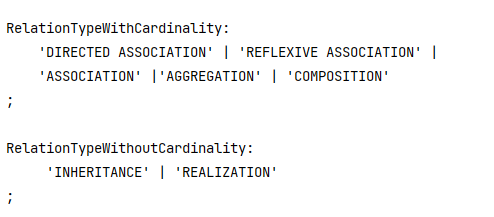
*Slika 4. Pravilo za navođenje parametara funkcije*

Kako su i atributi i funkcije vezani za tipove podataka, ova gramatika propisuje i koji tipovi su dozvoljeni. Na slici 5. Može se videti koji su onsovni tipovi podržani i na koji način je podržana upotreba složenih tipova kao što su lista i rečnik. Za povratnu vrednost funkcije dodat je *void* tip.



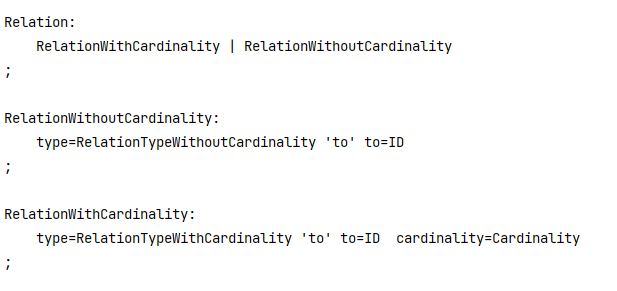
*Slika 5. Dozvoljeni tipovi podataka*

Pravila koja se tiču definisanja veza između elemenata dijagrama podeljena su na pravila za veze koje sadrže kardinalitet i na pravila za veze koje ne sadrže kardinalitet. Da bi to bilo moguće, prvobitno su napravljene dve kategorije veza. Podelu možemo videti na slici 6.

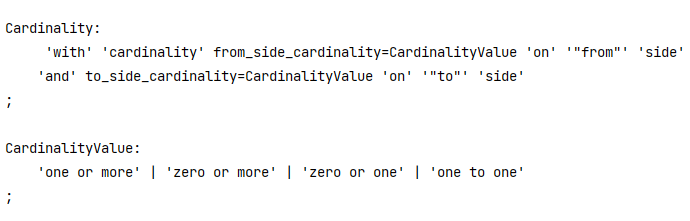


*Slika 6. Podela veza između elemenata dijagrama klasa*

Za veze uz koje se navodi kardinalitet, definisano je pravilo kako to učiniti. Takođe, i ovom prilikom se vodilo računa da navođenje kardinaliteta u velikoj meri liči na prirodni jezik, ali tako da ne bude predugo jer bi to dovelo do težeg čitanja jezika. Na slici 7 može se videti pravila za definisanje veza sa i bez kardinaliteta, a na slici 8 može se videti kako izgleda pravilo za definisanje kardinaliteta i koji kardinaliteti su dozvoljeni. Primer ispravno definisane veze sa kardinalitetom je: *DIRECTED ASSOCIATION to Address with cardinality zero or more on "from" side and zero or one on "to" side*

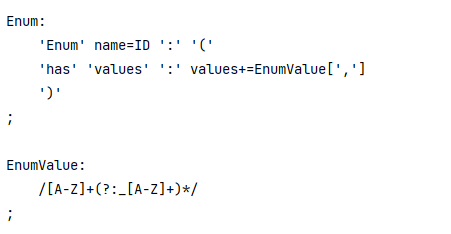


*Slika 7. Pravila za definisanje veza sa i bez kardinaliteta*



*Slika 8. Pravilo za definisanje kardinaliteta i dozvoljeni kardinaliteti*

Enumeracije su važan deo dijagrama klasa zato i ova gramatika obezbeđuje pravilo za njihovo definisanje. Na slici 9 se može videti na koji način se definišu enumeracije i njihove vrednosti. Vrednosti enumeracija moraju biti stringovi koji se sastoje od velikih slova i donje crte.



*Slika 9. Pravilo za definisanje enumeracije i njenih vrednosti*

Primeri fajlova koji sadže tekst napisan tako da odgovara ovoj gramatici DSL-a za dijagram klasa nalaze se u paketu *tests* i zovu se *correct\_file.md* i *correct\_file2.md*.

***Semantika jezika***

Semantika jezika formirana je tako da odgovara pravilima koja inače važu u dijagramu klasa sa manjim akcentom na pravilima koja se tiču veza između elemenata. Semantičke provere vrše se u fajlovima *custom\_classes.py* i *semantic\_check.py*. U fajlu *custom\_classes.py* proverava se jedinstvenost naziva klasa, interfejsa, enumeracija, atributa, parametara i potpisa funkcija.

***Implementacija provere jedinstvenosti naziva***

* Formiraju se zasebne liste za atribute i funkcije koje se prazne na kraju formiranja krajnjeg elementa kom pripadaju (klasi ili interfejsu)
* Lista atributa se popunjava nazivima atributa i ukoliko se pokuša dodavanje atributa čije ime se već nalazi u listi, program će baciti *AttributeAlreadyDeclared* exception.
* Lista funkcija se popunjava potpisima funkcija koji su u obliku „*naziv\_funkcije | povratna\_vrednost | broj\_parametara*“. Ukoliko se pokuša dodavanje funkcije čiji potpis se već nalazi u listi, program će baciti *FunctionAlreadyDeclared* exception. Prilikom prepoznavanja funkcija, prepoznaju se i njihovi parametri. Provera jedinstvenosti naziva parametara se obavlja tako što je lista parametara u okviru klase *Function* inicijalno prazna i prolaskom kroz prepoznate parametre, jedan po jedan se dodaje u listu ukoliko prethodno već nije dodat parametar sa istim nazivom. Ukoliko je prilikom navođenja parametara funkcije naznačeno da funkcija nema parametre (upotrebom reči *no*), a parametri su ipak navedeni, ispisaće se upozorenje o ovoj situaciji i da su parametri ipak sačuvani.
* Jedinstvenost naziva enumeracija, klasa i interfejsa se proverava tako što postoje zasebne mape za ove tri vrste elemenata gde su ključevi predstavljaju njihove nazive, a vrednosti su instance prepoznatih elemenata. Prilikom prepoznavanja nekog od elemenata, prethodno se u sve tri mape proverava da li naziv već postoji, ako postoji program vraća *ElementAlreadyDeclared* exception. Navedene mape se ujedno koriste kao tabele simbola za kasnije semantičke provere
* Provera jedinstvenosti vrednosti enumeracije obavlja se na isti način kao i provera jedinstvenog naziva parametara funkcije.

***Implementacija preostalih semantičkih provera***

Preostale semantičke provere implementirane su u fajlu *semantic\_check.py*. Provere se izvršavaju tako što se iteracijom kroz ključeve svake od mapa tj. tabele simbola (za enumeracije, klase i interfejse) dobavlja konkretan element i vrši se dodatna provera tipova atributa, parametara i povratnih vrednosti funkcija kao i provera određenih pravila koja se tiču veza sa drugim elementima dijagrama. Provera tipova podataka i veza se obavlja nakon celokupnog parsiranja fajla kako u jeziku ne bi postojalo ograničenje da je važan redosled navođenja elemenata (npr. da klasa 1 mora biti navedena pre klase 2 kako bi mogla postojati veza od klase 2 ka klasi 1).

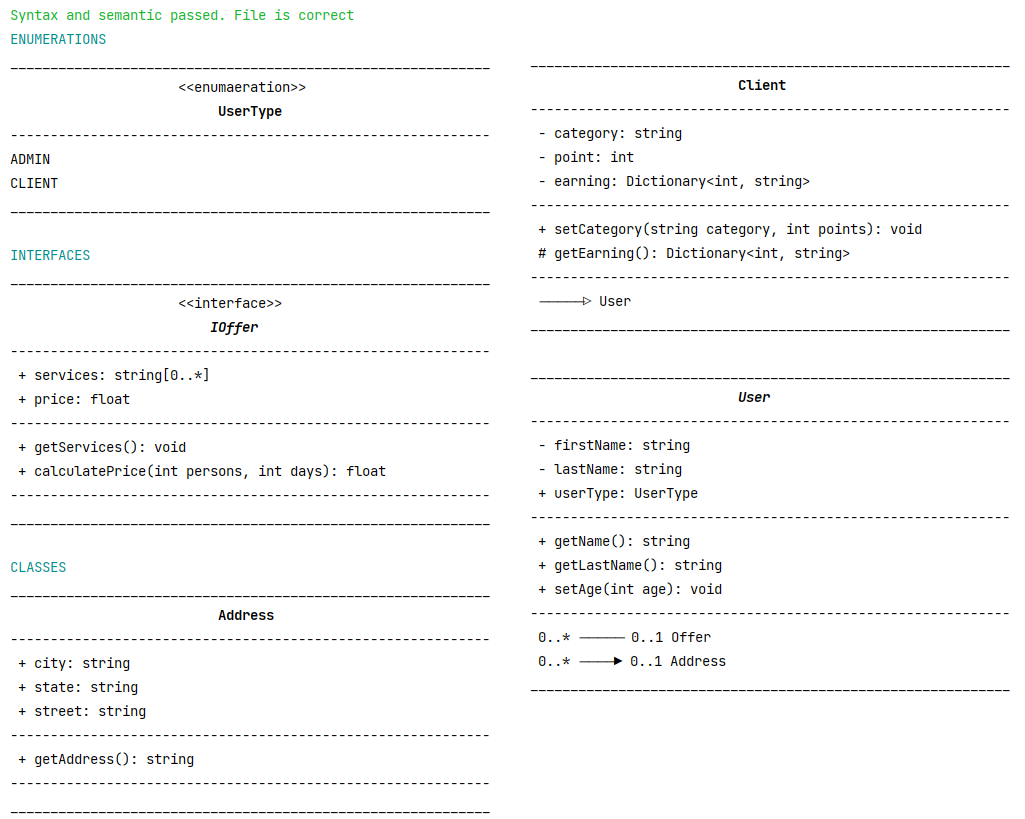
*Konkretne semantičke provere koje su implementirane:*

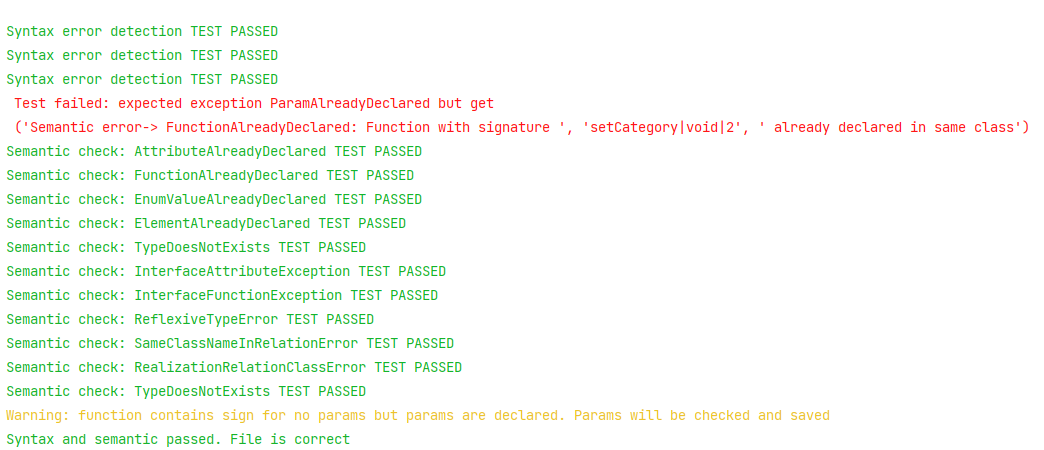
* + Scope atributa i funkcija interfejsa moraju biti public
  + Tip podataka koji ne spada u obične tipove i nije lista ili rečnik, mora biti definisan kao enumeracija, klasa ili interfejs u konkretnom dijagramu klasa
  + Element ka kom je usmerena veze od strane drugog elementa dijagrama mora biti definisan kao enumeracija, klasa ili interfejs u konkretnom dijagramu klasa
  + Veza tipa *reflexive association* mora sa obe strane imati isti element konkretnog dijagrama klasa
  + Ukoliko veza nije tipa *reflexive association*, ne može sa obe strane veze biti naveden isti element konkretnog dijagrama klasa
  + Veza tipa *realization* mora biti usmerena ka elementu dijagrama tipa interfjes
  + Ako veza veza tipa *inheritance* polazi od elementa tip interfejs, onda mora biti usmerena takođe ka elementu tipa interfejs

***Testiranje***

U projektu je implementirano automatsko pokretanje testnih fajlova prilikom pokretanja projekta odnosno *main.py* fajla. Testni fajlovi se nalaze u paketu tests i ima ih 17 ukupno. Od toga su 2 fajla u potpunosti u skladu za sintaksom i semantikom jezika (*correct\_file.md* i *correct\_file2.md*) dok su preostalih 15 primeri koji proveravaju detektovanje sintaksnih i semantičkih greški. Tri fajla pokazuju proveru sintakse i detekciju sintaksnih greški, dok su preostali fajlovi formirani tako da svaki pokazuje po jednu vrstu semantičke greške. Za svaku vrstu semantičke greške formirana je klasa koja nasleđuje Exception klasu i ispisuje poruku o grešci. Klase za exception-e nalaze se u fajlu *exceptions.py* u paketu *custom\_exceptions*.

Ukupno postoji 17 testovi i svi se pokreću odjednom. U slučaju fajlova koji su u skladu sa sintaksom i semantikom jezika, test je uspešan ako se ne detektuje ni jedan exception, a nakon testa se u konzoli iscrtava svaki element dijagrama kao što je prikazano na slici 10. U slučaju svih ostalih fajlova, test je uspešan ako se detektuje tačno onaj exception koji je predviđen. U suprotnom, ispisuje se koji exception je očekivan, a koji je dobijen. Primer jednog palog testa i ostalih uspešnih može se videti na slici 11. Na slici 11 se takođe može videti upozorenje koje se odnosi na upotrebu reči no prilikom navođenja parametara funkcije i istovremeno navođenje parametara.

*Slika 10. Uspešan test fajla koji je u skladu sa sintaksom i semantikom jezika i iscrtavanje dijagram*

*Slika 11. Uspešan prolaz svih testova osim jednog*

## Ideje za nastavak

Ideja koja je postojala na početku projekta je sprovedena i realizovana do kraja, tako da sa tog stanovišta nema elemenata projekta koji nedostaju. Međutim, kao što je na početku dela o sematičkim proverama navedeno, nije se do detalja vršila provera svih realnih pravila koja se tiču međusobnih veza elemenata u dijagramu klasa. Dodatno istraživanje pravila koja inače važe u dijagramu klasa bi doprinelo poboljšanju projekta i povećalo njegovu verodostojnost. Gramatika jezika mogla bi se tada prošiti novim pravilima i potencijalno još približila prirodnom jeziku. Takođe, proširenje je uvek moguće u pogledu testiranja formiranjem novih testnih fajlova koji bi pokazali ispravnost semantičkih i sintaksnih provera.

## Literatura

1. <http://textx.github.io/textX/3.0/?fbclid=IwAR2MpUP-UkQ6WMW0av1y_0WzznXsFBvU4rVtqOC3wHZDEqaKlqww14un0FQ>
2. <https://www.igordejanovic.net/courses/tech/textX/index.html?fbclid=IwAR0464klptDabrQAVzhGJFRiqYTAz-EDWWYR5GrEfwhn3TIVMGCSLXNBqfs>
3. <https://github.com/textX/textX>