Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/01BE15Elero

SERVER I EKSTENZIJA ZA VS CODE OKRUŽENJE ZA PODRŠKU JEZICIMA BAZIRANIM NA TEXTX ALATU

LANGUAGE SERVER AND VS CODE EXTENSION FOR DOMAIN SPECIFIC LANGUAGES BASED ON TEXTX

Daniel Elero, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu implementirana je ekstenzija za Visual Studio Code okruženje koja jezicima baziranim na alatu textX omogućava: prijavljivanje grešaka, dopunu koda, skok na definiciju pravila, prikaz svih referenci i code lens. Ekstenziju čine dve komponente: server i klijent, a njihova komunikacija odvija se preko Language Server Protocol-a.*

Ključne reči: *ekstenzija, textX, DSL, Language Server Protocol, Visual Studio Code, dopuna koda, prijavljivanje grešaka, navigacija do definicije pravila, prikaz svih referenci*

Abstract – *This paper presents implementation of a Visual Studio Code extension which provides linting, code completion, go to reference, find all references and code lens for domain specific languages based on textX. Extension has two components: language server and client which uses Language Server Protocol for their communication.*

Keywords: *extension, textX, DSL, Language Server Protocol, Visual Studio Code, code completion, linting, go-to definition, find all references*

1. UVOD

Ovaj rad obuhvata dizajn i implementaciju ekstenzije za VS Code okruženje koja jezicima definisanim u alatu textX nudi prijavljivanje grešaka, dopunu koda, „skok“ na definiciju pravila, prikaz svih referenci pravila i *code lens*. Ekstenzija se sastoji iz dve komponente: servera i klijenta koji podržavaju i komuniciraju preko *Language Server Protocol*-a (LSP). Ideja LSP-a jeste razdvajanje implementacione logike za pružanje „pametnih“ funkcionalnosti od njihovog prikazivanja i upotrebe u nekom od editora. Na ovaj način, svi editori koji podržavaju LSP (VS Code, Atom, Vim i dr.), vrlo lako se mogu integrisati sa serverom.

Jezici specifični za domen (DSL – *Domain Specific Language*) predstavljaju programske jezike ograničene ekspresivnosti, fokusirane na određenu oblast [1]. DSL-ovi imaju brojne prednosti u odnosu na jezike opšte namene:

- **Povećana produktivnost** programera, rezultat je velike ekspresivnosti i konciznosti jezika specifičnog za domen. Postoje studije koje pokazuju da produktivnost može narasti i 1000%
- **Kvalitetniji izvorni kod** rezultat je smanjenja broja linija programskog koda potrebnih da se reši problem.
- **Duži životni vek aplikacije** ogleda se u tome da je rešenje na višem nivou apstrakcije, odnosno, implementacija rešenja ne zavisi od izbora platforme i tehnologija
- **Angažman domenskih eksperata** omogućen je dizajnom jezika koji isključivo sadrži koncepte iz oblasti za koju je namenjen. Domenski eksperti često samostalno mogu da koriste ove tipove jezika.
- **Samodokumentujući** - zbog upotrebe koncepata iz domena

Martin Fovler u [1], deli DSL-ove u 3 kategorije: **eksterni**, **interni** i **jezičke radionice**.

Interni DSL nastaje proširivanjem jezika opšte namene, najčešće kreiranjem biblioteka. Ovakav pristup omogućava jednostavniju implementaciju i korišćenje integrisanih razvojnih okruženja (IDE) jezika na kome je zasnovan, ali na uštrb sintaksne ograničenosti.

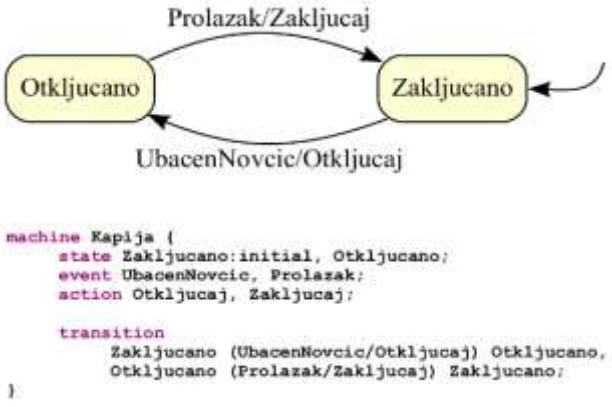
Eksterni DSL ima posebno definisanu sintaksu, kao i svoj interpreter ili kompajler. Semantika ovih tipova jezika dobija se prevođenjem na neki od jezika opšte namene ili njihovom interpretacijom.

Fovler u [2] uvodi termin jezičke radionice koja predstavlja okruženje za razvoj, testiranje i evoluciju jezika i propratnih alata. Najpoznatiji pripadnici ove grupe su Xtext i MPS.

Nezavisno od kategorije, svaki jezik specifičan za domen sastoji se od jedne ili više **konkretnih sintaksi**, **apstraktne sintakse** i **semantike.**
Konkretna sintaksa predstavlja vizuelnu reprezentaciju jezika preko koje korisnik vrši interakciju sa računarom i definiše izgled iskaza na nekom jeziku [3]. Ona može biti tekstualna, grafička, tabelarna, hibridna (kombinacija više vrsta), itd. Pri parsiranju tekstualne sintakse, kreira se stablo konkretne sintakse (eng. *Concrete Syntax Tree* - CST). Na slici 1 prikazane su tekstualna i grafička sintaksa koje imaju isto značenje.
Apstraktnu sintaksu čine koncepti domena, njihove osobine i korelacije. Ona predstavlja metamodel i služi za validaciju iskaza. Stablo apstraktne sintakse (eng.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Igor Dejanović, vanr. prof.

Abstract Syntax Tree - AST), takođe, nastaje parsiranjem ulaznog teksta, ali sadrži manje informacija od CST.



Slika 1 Grafička i tekstualna konkretna sintaksa [4]

Apstraktni semantički graf (*Abstract Semantic Graph* - *ASG*) predstavlja (uglavnom) direktan acikličan graf i na višem je nivou apstrakcije od AST.
Semantika definiše smisao jezičkih iskaza [4]. Takođe, semantiku jezika definiše skup ograničenja.

S obzirom da je tema rada usko vezana za alat textX, fokus rada usmeren je ka parserima i eksternim DSL-ovima sa tekstualnom konkretnom sintaksom.
Parseri su softverske komponente koje od ulaznog niza znakova i simbola, kreiraju strukture pogodne za dalju obradu (najčešće CST ili AST), a na osnovu formalno definisane gramatike. Za izdvajanje tokena iz ulaznog teksta, najčešće se koristi odvojen leksički analizator, ali postoje i parseri koji vrše tokenizaciju i parsiranje u jednom koraku (eng. *scannerless parsers*). Parseri mogu biti automatski ili polu-automatski generisani parser generatorima (eng. *parser generators*) [5].
Parsiranje se najčešće odvija u tri koraka [5]:

1. **Leksička analiza** vrši kreiranje tokena iz ulaznog teksta.
2. **Sintaksna analiza** vrši validaciju i proveru redosleda tokena u zavisnosti od definisane gramatike i kreira stablo parsiranja.
3. **Semantička analiza** daje značenje parsiranom ulazu, interpretirajući ga ili prevodeći na jezik razumljiv računaru. U ovoj fazi se, nad čvorovima stabla parsiranja, izvršavaju semantičke akcije.

Prema načinu parsiranja, Aho u [6] deli parsere na dve grupe:

- *top-down*
- *bottom-up*

Bottom-up **parseriranje** počinje identifikacijom terminalnih simbola analizirajući ulazne tokene sleva na desno. Terminali ujedno postaju i listovi stabla parsiranja. Nakon toga se, kombinacijom terminala, a u skladu sa produkcionim pravilima gramatike, grade čvorovi stabla koji predstavljaju neterminale. Ovaj proces se ponavlja dok se ne stigne do korenskog pravila (korena stabla parsiranja).

Top-down **parseri** kreću od korena stabla i kreiraju neterminale (čvorove) primenjujući redukciona pravila gramatike. Parsiranje traje sve dok se svi neterminali ne zamene terminalima (listovi stabla).

Metamodel čini specifikacija koncepata, korelacija i ograničenja potrebnih za konstrukciju modela. Proces kreiranja metamodela naziva se metamodelovanje [7]. Metamodeli su specificirani za određeni domen, odnosno služe za kreiranje modela za navedeni domen. Ako se podignemo za još jedan nivo apstrakcije, potreban nam je formalizam za opis metamodela, a to je meta-metamodel. Domen za koji je meta-metamodel namenjen jeste kreiranje metamodela.

2. PREGLED STANJA IZ OBLASTI

2.1 Arpeggio

Arpeggio [8] predstavlja opadajući rekurzivni parser sa *backtracking*-om i memoizacijom (tzv. *pacrat parser*). Razvijen je u programskom jeziku Python od strane profesora Igora Dejanovića na Fakultetu Tehničkih Nauka u Novom Sadu. Gramatika jezika zadaje se PEG notacijom (slika 2) ili Python funkcijama (slika 3), a, za razliku od parser generatora, Arpeggio se konfiguriše „u letu".

```
robot = 'begin' command* 'end' EOF
command = UP/DOWN/LEFT/RIGHT
UP = 'up'
DOWN = 'down'
LEFT = 'left'
RIGHT = 'right'
```

Slika 2 Primer gramatike definisane PEG notacijom

```
def robot():    return 'begin', ZeroOrMore(command), 'end', EOF
def command():  return [UP, DOWN, LEFT, RIGHT]
def UP():       return 'up'
def DOWN():     return 'down'
def LEFT():     return 'left'
def RIGHT():    return 'right'
```

Slika 3 Primer gramatike definisane Python funckijama

Parsiranjem odgovarajućeg modela dobijamo AST i stablo parsiranja nad kojim je moguće definisati semantičke akcije. Definisanje semantičkih akcija vrši se nasleđivanjem klase ***PTNodeVisitor***, u okviru koje se navode metode oblika ***visit_*** iza čega sledi ime pravila. Pored samog parsiranja, Arpeggio ima podršku za debagovanje i vizualizaciju modela i stabla parsiranja.

2.2 Xtext

XText je razvojni okvir namenjen za kreiranje tekstualnih DSL-ova. Za opis gramatika koristi se metajezik Xtext, koji sadrži predefinisane tipove, anotacije, enumeracije, a dozvoljeno je i kombinovanje više gramatika. Pored opisa gramatike, ovaj razvojni okvir omogućava dodavanje različitih semantičkih validacija, prilagođavanje i modifikacije prikaza strukture koda, dopune koda i ostalih karakteristika editora kao i pisanje generatora za mapiranje modela na, najčešće, jezike opšte namene.

XText metajezik je poslužio kao inspiracija za kreiranje alata textX, a skup alata vezanih za generisanje i funkcionisanje *Eclipse* editora kreirao je ideju o proširenju textX-a u istom pravcu, što je istraženo i implementirano kroz ovaj master rad.

2.3 textX

TextX [9] je alat namenjen brzom kreiranju eksternih jezika specifičnih za domen. Implementiran je u programskom jeziku Python na Fakultetu tehničkih nauka u Novom Sadu od strane profesora Igora Dejanovića, a moguće ga je instalirati pip komandom na sledeći način: *pip install textX*. TextX ujedno predstavlja i metajezik čija je notacija veoma slična XText-ovoj, dok u pozadini koristi Arpeggio parser.

Za razliku od XText-a koji je poslužio kao inspiracija za kreiranje ovog alata, a zahvaljujući dinamičkoj prirodi jezika Python, textX ima sposobnost da "u letu" konfiguriše Arpeggio, kreira metamodel i parsira tekstualne modele koji odgovaraju specificiranoj gramatici.

Metamodel čine Python klase koje odgovaraju pravilima gramatike i sadrže njihove relevantne informacije, a proces parsiranja ulaznog modela kreira objektni graf čiji čvorovi su instance klasa metamodela.

Kreirani objektni graf je dalje moguće interpretirati, ili iz njega generisati izvorni kod na nekom od jezika opšte namene.

Čuvajući glavnu karakteristiku textX-a, ideja ovog master rada je da "u letu", na osnovu gramatike i dodatnih specifikacija, prilagodi jezički server DSL-u.

2.4 *Language server protocol*

Veliki broj IDE-a i tekstualnih editora za upravljanje izvornim kodom, sa jedne strane, i sve veći broj programskih jezika i njihov aktivni razvoj, sa druge strane, povećava resurse potrebne za razvoj i održavanje alata za „pametnu" podršku tim jezicima. LSP je baziran na JSON RPC (*Remote Procedure Call*) protokolu i specificira skup metoda i notifikacija za komunikaciju između servera za podršku određenom jeziku i različitih klijenata (IDE-a) i time rešava navedeni problem.

3. IMPLEMENTACIJA

Ekstenziju čine server i klijent. Implementacija klijenta je trivijalna i sadrži samo logiku potrebnu za pokretanje i uspostavljanje komunikacije sa serverom ukoliko se desi jedan od sledećih događaja:

- Kreiranje ***.txconfig*** fajla u okviru radnog prostora
- Otvaranje fajlova sa ekstenzijom ***.tx***

Fajl ***.txconfig*** (slika 4) predstavlja konfiguracioni fajl koji sadrži informacije kao što su: ekstenzije fajlova ciljanog DSL-a, putanja do gramatike jezika, putanje do ostalih konfiguracionih fajlova i dr.

Nakon konfiguracije i aktivacije, server je spreman za korišćenje.

```
dsl MyDSL [mydsl, my] {
    general {
        author: "Daniel Elero"
        version: "1.0.0"
    }
    paths {
        grammar: "./mydsl/grammar.tx"
        classes: "./mydsl/lang.py:get_classes"
        builtins: "./mydsl/lang.py:get_builtins"
        model_processors: "./mydsl/lang.py:get_model_proc"
        object_processors: "./mydsl/lang.py:get_obj_proc"
        match_filters: "./mydsl/lang.py:get_filters"
    }
}
```

Slika 4 Konfiguracioni fajl (***.txconfig***)

3.1 „Skok" na definiciju pravila

"Skok" na definiciju pravila (eng. ***go to definition***) podrazumeva da se preko imena instance pravila, "skoči" na njegovu definiciju u okviru istog ili drugog fajla. Ova funkcionalnost nam omogućava brzo kretanje po izvornom kodu u toku razvoja, ili za vreme pregleda koda. Implementacija opisane funkcionalnosti zahtevala je i proširenje alata textX kome je dodato polje za čuvanje i lak pristup svim pravilima na osnovu njihove pozicije. Korišćenjem binarne pretrage, algoritam na osnovu trenutne pozicije kursora u editoru, pretražuje pravila i, ukoliko je pravilo pronađeno, vraća fajl i poziciju na kojoj se definicija nalazi.

3.2 Pretraživanje referenci

Pretraživanje i pregled referenci (eng. ***find all references***) predstavlja inverznu funkcionalnost u odnosu na „skok" do definicije pravila, a pruža nam uvid koliko puta i gde se instanca nekog pravila koristila u modelu. Funkcija koja vrši pretragu svih referenci pravila sadrži sledeće korake:

1. Za trenutnu poziciju kursora u editoru, dobavlja se instanca pravila
2. Iz liste svih referenciranih pravila u modelu, biraju se ona pravila koja se poklapaju sa pravilom dobijenim u koraku 1
3. Dobijena lista se mapira na objekte koji odgovaraju LSP specifikaciji

3.3 Prijavljivanje grešaka

Prijavljivanje grešaka tokom pisanja koda predstavlja jednu od najznačajnijih funkcionalnosti ekstenzije. Implementacija se oslanja na alate textX i Arpeggio i njihove mehanizme hvatanja izuzetaka. Na svaku promenu teksta ili fajla, server proverava trenutno stanje modela na osnovu zadate gramatike i putem notifikacija obaveštava klijenta o greškama.

3.4 Dopuna koda

Dopuna koda (eng. ***code completion***) je funkcionalnost koju klijent poziva da bi, na osnovu trenutne pozicije kursora u tekstu, dobio predlog o potencijalnim tokenima koji mogu da se nađu na tom mestu. Trenutna implementacija oslanja se na textX *i* Arpeggio, odnosno, za listu stavki dopune koda, predlažu se tokeni koje parser zahteva na datom mestu.

Algoritam za kreiranje liste predloga izvršava se u sledećim koracima:

1. Klijent šalje zahtev za dopunu koda
2. Server na poziciji kursora ubacuje karaktere koji će model učiniti nevalidnim
3. textX preko Arpeggio parsera dobija moguće validne tokene na mestu greške
4. U zavisnosti od toga da li je greška sintaksna ili semantička, textX dopunjuje informacije o grešci
5. Greška sa listom predloga se vraća do servera
6. Server filtrira i proširuje listu predloga i vraća ih klijentu
7. Predlozi se prikazuju u okviru korisničkog interfejsa

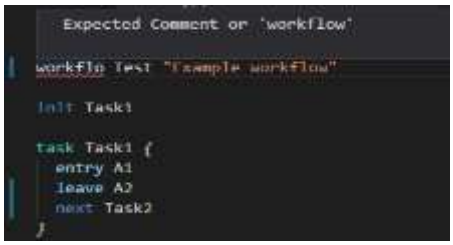
3.5 Generisanje ekstenzije

Generisanje ekstenzije predstavlja funkcionalnost koja na osnovu konfiguracionih fajlova vrši kreiranje i pakovanje ekstenzije specifične za određeni domenski jezik i omogućava njihovu distribuciju.

4. VERIFIKACIJA I UPOTREBA EKSTENZIJE

Za verifikaciju ekstenzije odabran je jezik *Workflow* čija je namena definisanje aktivnosti, akcija, i zadataka za opis proizvoljnog posla. Nakon kreiranja gramatike i konfiguracionog fajla, izgenerisana je namenska ekstenzija sa sledećim funkcionalnostima:

Prijavljivanje grešaka (slika 6), predstavljeno je podvlačenjem mesta na kome se nalazi greška.

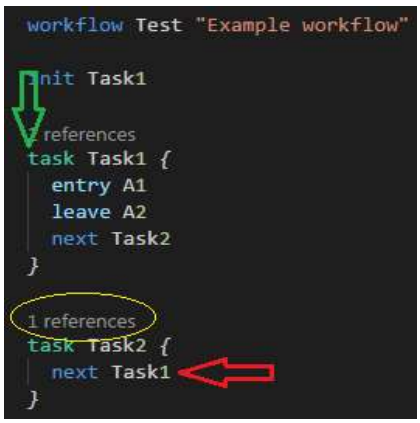


Slika 6 Prikaz sintaksne greške

Dopuna koda (slika 7) predstavlja prikaz validnih tokena na trenutnoj poziciji kursora.

Slika 7 Prikaz dopune koda

„Skok“ na definiciju pravila i **prikaz svih referenci** povezane su funkcionalnosti i prikazane su na slici 8. Kombinacijom tastera CTRL i pritiskom levog klika miša na *Task1* (crvena strelica), kursor se prebacuje na početak njegove definicije (zelena strelica), dok žuta elipsa naznačuje koliko je puta neko pravilo referencirano u modelu.



Slika 8 „Skok“ na definiciju pravila i prikaz svih referenci

5. ZAKLJUČAK

Cilj ovog master rada bio je istraživanje postojećih rešenja iz oblasti parsera, metajezika, namenskih IDE-a i implementacija generičkog jezičkog servera za podršku jezicima definisanim u alatu textX. Karakteristika implementiranog jezičkog servera da se dinamički prilagođava prema metamodelu jezika i dodatnim specifikacionim fajlovima čuva prednosti textX-a u odnosu na konkurentske alate.

Implementirana ekstenzija pruža prijavljivanje grešaka, dopunu koda, „skok“ na definiciju pravila, pretraživanje referenci pravila i *code lens*.

LITERATURA

[1] M. Fowler, “Domain specific languages.”, Addison-Wesley Professional, September 24, 2010.

[2] M. Fowler, “Language Workbenches: The Killer-App for Domain Specific Languages?” Dostupno na http://www.issi.uned.es/doctorado/generative/Bibliografia/Fowler.pdf. Pristupano: Avgust 2018.

[3] M. Völter, “DSL Engineering”, 2010-2013.

[4] http://www.igordejanovic.net/courses/jsd/uvod.html. Pristupano: Avgust 2018.

[5] https://en.wikipedia.org/wiki/Parsing. Pristupano: Avgust 2018.

[6] A. V. Aho, M. S. Lam , R. Sethi, J. D. Ullman, “Compilers : principles, techniques, and tools” - 2nd ed.Pearson Education, 2007

[7] I. Dejanović, “Prilog metodama brzog razvoja softvera na bazi proširivih jezičkih specifikacija. PhD thesis, Faculty of Technical Sciences, University of Novi Sad, January 2012”

[8] I. Dejanović, B. Perišić, G. Milosavljević, “ARPEGGIO: Pacrat parser interpreter”, Fakultet tehničkih nauka, Univerzitet u Novom Sadu, 2010.

[9] I. Dejanović, R. Vaderna, G. Milosavljević, Ž. Vuković, “textX: A Python tool for Domain-Specific Languages Implementation”. Faculty of Technical Sciences, University of Novi Sad, 2016.

Kratka biografija:

Daniel Elero rođen je u Novom Sadu 1993. godine. Master rad na Fakultetu tehničkih nauka u Novom Sadu, iz oblasti Elektrotehnike i računarstva – softversko inženjerstvo, odbranio je 2018. godine.