Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/01BE14Kupco

PODRŠKA VIZUALIZACIJI JEZIKA KREIRANIH UPOTREBOM TEXTX BIBLIOTEKE U OKVIRU VISUAL STUDIO CODE EDITORA

LANGUAGE VISUALIZATION SUPPORT FOR TEXTX-BASED LANGUAGES IN VISUAL STUDIO CODE EDITOR

Daniel Kupčo, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Ovaj rad predstavlja implementaciju podrške za vizualizaciju bilo kog DSL-a (eng. Domain Specific Language) kreiranog pomoću textX biblioteke. Realizovana je kao ekstenzija za Visual Studio Code editor u kombinaciji za posebno kreiranim DSL-om za opis vizualizacije pod nazivom viewX i različitim bibliotekama koje omogućavaju vizualizaciju modela.*

Ključne reči: *DSL, vizualizacija, textX, viewX, Visual Studio Code, ekstenzija*

Abstract – *This paper presents implementation of a support for the visualization of any textX-based DSL (Domain Specific Language). It is realized as an extension for Visual Studio Code editor in combination with specially created DSL for visualization description called viewX and a variety of libraries which enable model visualization.*

Keywords: *DSL, visualization, textX, viewX, Visual Studio Code, extension*

1. UVOD

U ovom radu predstavljena je implementacija ekstenzije za *Visual Studio Code* editor, čija je uloga da omogući vizualizaciju jezika specifičnih za domen, ili kraće DSL-ova (*eng. Domain Specific Languages*) kreiranih upotrebom *textX* biblioteke.

Problematika se ogleda u vizualizaciji struktura podataka tipa grafa, što predstavlja zasebnu oblast izučavanja u matematici i računarskim naukama. Pored toga, rešenje mora biti generičko, odnosno, potrebno je omogućiti vizualizaciju bilo kog modela jezika kreiranog pomoću *textX* biblioteke, i to na takav način da se korisniku pruža mogućnost modifikovanja načina vizualizacije i samog izgleda grafa kao njenog rezultata.

Značaj alata koji se bave problemom vizualizacije ogleda se u činjenici da korisnicima omogućavaju lakše i bolje razumevanje semantike napisanih modela, oslanjajući se na vizualne elemente i metode prikazivanja. Dodatno, vizualizacija programerima može da olakša uočavanje grešaka unutar modela čime se smanjuje vreme potrebno za njihovo ispravljanje. Vizualno prikazivanje takođe može da omogući i bolju komunikaciju između programera i domenskih eksperata jer je moguće predstaviti apstrakciju modela na takav način da se ključna svojstva i relacije koje su definisane unutar njega mogu jasno uočiti i interpretirati. Jedna od prednosti implementacije koja je predstavljena u ovom radu jeste mogućnost vizualizacije modela bilo kog jezika kreiranog pomoću *textX* biblioteke. Druga prednost koja se može izvdvojiti jeste izolacija načina vizualizacije od njenog sadržaja; sadržaj je deo samog modela koji se prikazuje, a način na koji se prikazuje definisan je *viewX* modelom što implicitno omogućava da se jedan model može prikazati na više različitih načina u zavisnosti od potrebe, a sa druge strane, više modela se mogu prikazati na isti način.

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

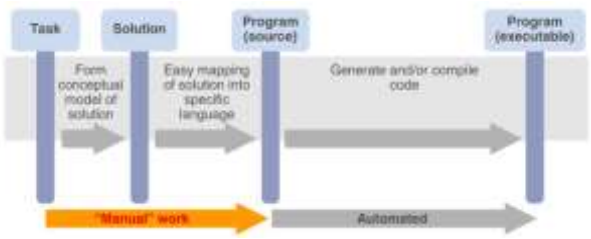
2. TEORIJSKE OSNOVE

2.1. Grafovi

Strukturu modela nekog jezika najpogodnije je predstaviti strukturom tipa grafa. Graf omogućava jednostavan prikaz koncepata definisanih u modelu i njihovih međusobnih relacija jer se koncepti mogu predstaviti čvorovima grafa, a njihove međusobne relacije kao veze između čvorova. Najveći izazov prilikom vizualizacije grafova predstavlja odabir pogodnog algoritma za raspoređivanje čvorova koji će zadovoljiti potrebne estetičke kriterijume.

2.2. Jezici specifični za domen

Jezici specifični za domen predstavljaju posebno kreirane jezike namenjene za upotrebu u okviru usko specifičnog domena. Konstruktivni elementi jezika predstavljeni su konceptima iz domena koji omogućavaju da konceptualan model rešenja bude direktno preslikan na konkretan model DSL-a koristeći koncepte iz domena problema.



Slika 1. *Proces razvoja rešenja upotrebom DSL-a [1]*

Za razliku od modela napisanih u jezicima opšte namene, DSL modeli su jednostavniji za razumevanje, što doprinosi boljoj komunikaciji između programera i

domenskih eksperata. čime se značajno umanjuje vreme potrebno za razvoj rešenja.

3. KORIŠĆENE TEHNOLOGIJE I ALATI

3.1. Python, textX i Arpeggio

Deo ekstenzije koji se oslanja na upotrebu *textX* biblioteke realizovan je u *Python* okruženju. *TextX* predstavlja alat koji omogućava kreiranje jezika specifičnih za domen [2]. Pomoću *textX* meta-jezika definiše se gramatika jezika na osnovu koje se konstruiše *Arpeggio [3]* parser i meta-model koji, u suštini, predstavljaju kreiran DSL spreman za korišćenje.

3.2. JavaScript, TypeScript i Visual Studio Code

Visual Studio Code editor, kao i svi moderni editori izvornog koda, imaju modularnu arhitekturu i podržavaju koncept proširivosti putem ekstenzija [4]. Jezgro *ovog* editora se zasniva na *JavaScript* okruženju, odnosno pokreće ga *Node.js* server. Stoga, *Visual Studio Code* editor propisuje svoj interfejs proširivosti putem pisanja *TypeScript/JavaScript* ekstenzija.

4. IMPLEMENTACIJA

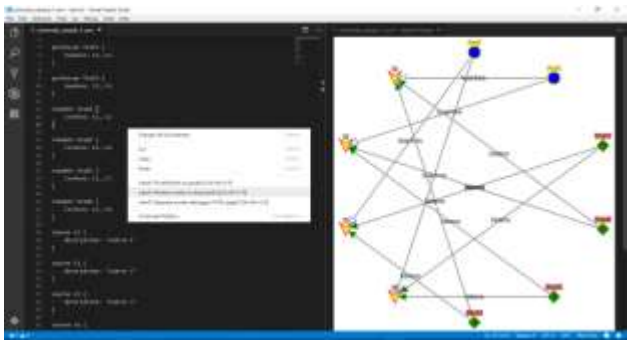
Implementacija ovog rada je realizovana kao ekstenzija za *Visual Studio Code* editor u kombinaciji sa posebno razvijenim *DSL*-om za opisivanje vizualizacije razvijenim pomoću *textX* biblioteke pod nazivom *viewX*. Iako je osnova ovog rešenja *Visual Studio Code* ekstenzija ono zapravo predstavlja integraciju nekoliko alata i biblioteka iz više okruženja.

4.1. viewX DSL

Sastavni deo ovog rada jeste i *viewX*, jezik kreiran upotrebom *textX* biblioteke. Njegova uloga je da opiše vizualizaciju modela nekog drugog *textX* baziranog DSL-a.
Kako bi se obezbedila što kompletnija kontrola vizualizacije, gramatika *ViewX* jezika kreirana je s ciljem da, pored osnovnih vizuelnih karakteristika elemenata modela, pruži i mogućnost definisanja izgleda grafa kao celine putem odabira različitih algoritama za raspoređivanje elemenata grafa.

4.2. viewX ekstenzija

Editor je pomoću ekstenzije proširen u vidu komandi i opcija putem kontekstnog menija kojima je omogućeno kreiranje novih projekata i interakcija sa ekstenzijom i mehanizmima vizualizacije unutar editora (Slika 1).



Slika 2. *Primer vizualizacije modela*

Bitan deo ekstenzije predstavlja i logika koja se bavi interpretacijom *viewX* modela koji opisuje elemente vizualizacije. Kako se ovaj rad bavi vizualizacijom modela jezika kreiranih pomoću *textX* biblioteke, ovaj deo ekstenzije je realizovan pomoću *Python* jezika jer je i sama *textX* biblioteka realizovana u okviru *Python* okruženja. Ovaj deo je ključan u procesu vizualizacije jer predstavlja vezu između konkretnog modela nekog jezika i grafa koji predstavlja rezultat vizualizacije. Ovaj graf se konstruiše tako što se najpre interpretira konkretan model jezika koji želimo da vizualizujemo a potom i *viewX* model. Kombinacijom ove dve interpretacije i na osnovu prikupljenih informacija iz oba modela kreira se model grafa koji predstavlja apstrakciju konkretnog modela koji se vizualizuje a koji odgovara parametrima definisanim u okviru *viewX* modela.

Polazna tačka i osnova u kojoj je smeštena logika za interpretaciju modela i konstruisanje modela grafa jeste *ViewXInterpreter.py* klasa. Uloga ove klase je ključna i ona obuhvata sve od interpretacije oba modela putem algoritama za prolazak kroz stabla objekata modela dobijenih nakon interpretacije, preko konstruisanja modela grafa sa svim informacijama vezanim za vizualna svojstva elemenata grafa, pa sve do generisanja modela grafa i propratnog koda u vidu HTML fajla, čijim će se zapravo učitavanjem graf vizualizovati. Iako je većina logike koncentrisana u ovoj klasi, stvari poput generisanja fajlova, pomoćnih metoda i adaptacije modela sa ciljnom bibliotekom za vizualizaciju su ipak smeštene u drugim *Python* skriptama zarad bolje organizacije koda.

4.3. Python-shell

Kako se ekstenzija izvršava u *JavaScript* okruženju, potrebno je nekako povezati ekstenziju sa kodom napisanim u *Python*-u. U ovu svrhu se koristi *python-shell* biblioteka [5]. Ova biblioteka omogućava izvršavanje *Python* skripti unutar novog procesa kreiranog u okviru *JavaScript* koda čime se omogućava komunikacija između ova dva okruženja. Ova komunikacija se zasniva na slanju podataka putem standardnih ulazno-izlaznih tokova podataka. Pozivanje koda unutar *Python* skripti putem ove biblioteke je vrlo jednostavno (Slika 2) a u slučaju nastalih grešaka u kodu koji se poziva podržan je i mehanizam za upravljanje izuzecima.

4.4. Jinja2

Kao što je već pomenuto, rezultat interpretacije modela jeste generisanje HTML fajla, čijim učitavanjem započinje vizualizacija modela, a koji sadrži opis strukture modela grafa zajedno sa *CSS* i *JavaScript* kodom koji obezbeđuju željeno stilizovanje i interakciju grafa. Ovaj kod adaptiran je, i obezbeđuje integraciju sa ciljnom bibliotekom za vizualizaciju grafa.

Generisanje ovog fajla se odvija korišćenjem *Jinja2* biblioteke. *Jinja2* predstavlja *Python* modul koji omogućava generisanje fajlova korišćenjem šablona (*eng. template*). Šablon omogućava generisanje fajlova tako što se dinamički sadržaj razdvaja od statičkog sadržaja blokovima koda koji će se potom generisati na osnovu prosleđenih *Python* objekata.

4.5. Cytoscape.js

Alat čija je uloga vizualizacija grafa jeste *Cytoscape.js* biblioteka. Ovo je *JavaScript* biblioteka zasnovana na teoriji grafova i namenjena je za analizu i vizualizaciju grafova. Postoji nekoliko biblioteka koje se bave vizualizacijom grafova i svaka je specifična na svoj način. Neke od osobina koje izdvajaju *Cytoscape.js* u od ostalih alata jesu mogućnost prilagođavanja putem mnogobrojnih parametara kao što su zumiranje, kretanje po grafu, raspored iscrtavanja čvorova grafa itd., zatim stilizovanja izgleda čvorova grafa i njihovih veza, kreiranje kompozitnih čvorova (čvorovi mogu sadržavati druge čvorove), potom izvršavanje različitih algoritama nad elementima grafa, animacije, mogućnost čuvanja različitih informacija unutar elemenata grafa kao i korišćenje tih informacija za različite potrebe.

Pored navedenih osobina, *Cytoscape.js* biblioteka podržava i proširivanje putem sopstvenih ekstenzija. Korisnik može, putem definisanog interfejsa, da proširi skup podržanih funkcionalnosti biblioteke u vidu dodatnih algoritama za raspoređivanje elemenata grafa, novih elemenata korisničkog interfejsa kao i funkcionalnih elemenata u vidu *clipboard* i *undo-redo* mehanizama. Takođe, podržan je i mehanizam reagovanja na događaje od strane korisnika. Određene akcije korisnika emituju događaje na koje je moguće registrovati metode koje će se pozvati kao reakcija na ove događaje. Ulazni parametar ovih metoda predstavlja objekat koji nosi osnovne informacije o nastalom događaju poput vremena nastanka, tip događaja, koordinate, element grafa ukoliko je nad njim načinjen itd. Implementacijom ovih metoda zapravo definišemo ponašanje kojim želimo da reagujemo na nastanak ovih događaja.

4.6. Socket.io

Budući da postoji mehanizam reagovanja na događaje i u okviru editora i na nivou grafa, postavlja se pitanje kako ove događaje propagirati sa jedne na drugu stranu i reagovati na njih, imajući u vidu da se ekstenzija i vizualizacija grafa izvršavaju unutar dva odvojena procesa. Odgovor na ovo pitanje donosi *Socket.io* biblioteka.

Socket.io je *JavaScript* biblioteka koja omogućava bidirekcionu komunikaciju u realnom vremenu između dva procesa oslanjajući se na mehanizam utičnica (*eng. socket*). Ova komunikacija je realizovana implementacijom *publish and subscribe* šablona nad klijent-server arhitekturom.

Navedena arhitektura je u implementaciji ovog rada primenjena tako da sama ekstenzija u okviru editora, kao i prozori unutar kojih se učitava generisani HTML fajl, a samim tim i vizualizuje graf koji je u njemu definisan, imaju ulogu klijenata. Svaki klijent, u zavisnosti od njegove uloge da li je to ekstenzija ili klijent u kome se vizualizuje graf, je registrovan na određene događaje a istovremeno, svaki od njih šalje određene događaje koji nastanu u njemu ostalim klijentima. Ovi događaji se prosleđuju *Socket.io* serveru koji ih potom prosleđuje svim drugim klijentima istovremeno. Klijenti sve vreme slušaju na događaje i, ukoliko se desi neki od njih na koji su oni registrovani, poziva se metoda koja je registrovana kao reakcija na te događaje. Ovim je omogućena obostrana komunikacija između ekstenzije i klijenata unutar kojih se vrši vizualizacija grafa.

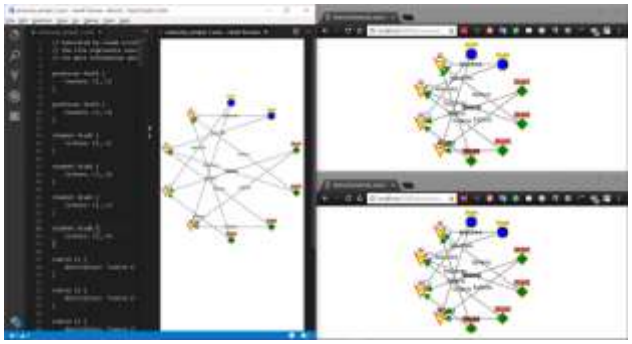
4.7. Browsersync

Svrha vizualizacije je da u svakom trenutku oslikava stvarno stanje modela pa se, iz ovog razloga, nakon sačuvanih izmena unutar modela iznova generiše HTML fajl sa novim modelom grafa koji reflektuje novonastale izmene. Problem koji se nameće jeste kako obavestiti sve klijente u kojima se graf vizualizuje da je došlo do promena i da se njihov prikaz treba osvežiti. U ovu svrhu koristi se *Browsersync*.

Browsersync je *JavaScript* biblioteka koja omogućava pristup statičkim fajlovima pute jedinstvene IP adrese. Svi klijenti koji vizualizuju graf, pristupaju fajlovima koji omogućavaju prikazivanje njegovog sadržaja pristupajući IP adresi putem *Browsersync* servera. *Browsersync* poseduje informaciju o svim klijentima koji su mu pristupili i, kada dode do izmene sadržaja ovih fajlova, *Browsersync* server javlja svakom klijentu da treba da osveži učitani sadržaj i svoje prikazivanje. Na ovaj način je omogućeno da svaki od aktivnih klijenata u svakom trenutku vizualizuje graf koji predstavlja verodostojni prikaz aktuelnog stanja modela.

4.8. Demonstracija ekstenzije

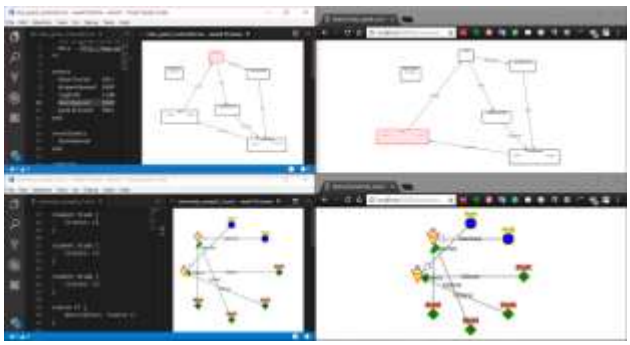
Iako je osnovni cilj ovog rada vizualizacija modela u okviru *Visual Studio Code* editora, ekstenzija poseduje nekoliko osobina i funkcionalnosti koje je izdvajaju od ostalih sličnih alata. Jedna od takvih funkcionalnosti je i mogućnost višestruke vizualizacije (Slika 3).



Slika 3. *Primer višestruke vizualizacije istog modela*

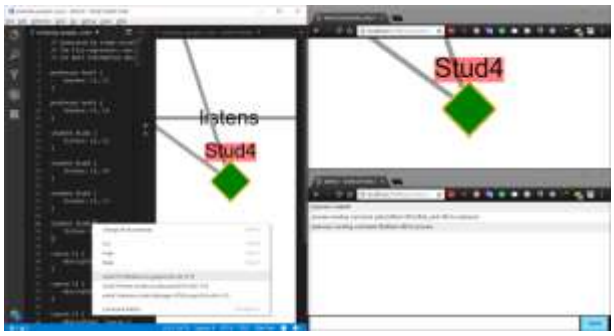
Svaka od ovih vizualizacija, bilo u okviru editora ili internet pretraživača, je potpuno nezavisna. Time je omogućeno da više korisnika istovremeno vizualizuju model i vrše interakciju nad grafom a da pri tome ne utiču na vizualizacije drugih korisnika.

Pored višestruke vizualizacije istog modela, moguće je i pokrenuti više instanci *Visual Studio Code* editora od kojih svaka može da istovremeno vizualizuje svoj model (Slika 4). Ovim je omogućena višestruka vizualizacija više modela što može da bude vrlo korisno kada je potrebno istovremeno analizirati nekoliko modela ako npr. želimo da uporedimo dva modela ili da prikažemo isti model na dva različita načina.



Slika 4. Višestruke *viewX* instance u paraleli

Interakcija sa grafom je ključna tokom analize i njegove interpretacije zarad boljeg razumevanja modela. Interakcija inicira događaje na koje ekstenzija reaguje, čime se uspotavlja komunikacija oslanjajući se na *Socket.io* server. Ukoliko se nešto ne dešava onako kako se očekuje, ne postoji mogućnost da se otkrije u kom momentu je došlo do problema u komunikaciji između delova ekstenzije.
Iz ovog razloga je vizualizacija dodatno podržana i sa mogućnošću otkrivanja grešaka, odnosno *debug* sesije (Slika 5). Ukoliko se *debug* opcija omogući za neku *viewX* instancu, moguće je otvoriti posebno kreiranu stranicu u okviru internet pretraživača u kome je u realnom vremenu moguće pratiti svaku poruku koja se šalje između klijenata i servera *Socket.io* sistema u oba smera.



Slika 5. *Demonstracija debug sesije*

Debug prozor, koji takođe predstavlja klijenta u okviru *Socket.io* klijent-server arhitekture, se nakon aktivacije i njegove konekcije sa *Socket.io* serverom registruje na sve poruke koje stižu do i odlaze od njega, te se njihov sadržaj prikazuje u glavnom delu prozora u hronološkom redosledu.

5. ZAKLJUČAK

Glavni cilj ovog rada bio je da se korisniku omogući vizualizacija modela nekog JSD-a kreiranog pomoću *textX* biblioteke ali na takav način da se nadomeste nedostaci sličnih alata za vizualizaciju modela koji su trenutno dostupni za upotrebu. Ekstenzija je od starta implementirana tako da omogućava generičnost, konfiguraciju i eventualna proširenja od strane korisnika kako bi odgovorila na njegove zahteve.

Jedna od najznačajnijih osobina *viewX* ekstenzije koja se izdvaja među ostalim rešenjima jeste generičnost. Može se posmatrati i kao nedostatak jer je za primenu potrebno da korisnik uloži više truda kako bi se generičnost konkretizovala, što smanjuje jednostavnost i brzinu upotrebe. Međutim, rezultat i sloboda dobijeni tom generičnošću nadmašuju uloženi napor.

Za dalji razvoj, u prvom redu, neophodno je obezbediti bolju integraciju sa *Cytoscape.js* bibliotekom i obogaćivanje *viewX* gramatike, čime će se obezbediti bolja kontrola nad generisanjem grafa i njegovom vizualizacijom, kao i mogućnost kasnije konfiguracije. Time bi se mogla omogućiti implementacija različitih funkcionalnosti poput boljeg raspoređivanja elemenata grafa, podršku za animaciju tokom vizualizacije, prikazivanje kompozitnih grafova (grafovi unutar grafova), reagovanje na različite događaje i još mnogo toga.

6. LITERATURA

[1] Sergey Dmitriev, JetBrains onBoard,“Language oriented programming: The next programming paradigm”, (2004)
[2] Dejanović Igor, Vaderna Renata, Milosavljević Gordana, Vuković Željko, “TextX: A Python tool for Domain-Specific Languages implementation”, Knowledge-Based Systems, vol. 115, 1-4, 2017.
[3] Igor Dejanović, Gordana Milosavljević, Renata Vaderna, „Arpeggio: A flexible PEG parser for Python“, Knowledge-Based Systems, vol. 95, 71-74, 2015
[4] Microsoft, “Extending Visual Studio Code”, Visual Studio Code, https://code.visualstudio.com/docs/extensions/overview (datum pristupa: 02.08.2018.)
[5] “Python-Shell”, GitHub, https://github.com/extrabacon/python-shell (datum pristupa: 02.08.2018.)

Kratka biografija:

Daniel Kupčo rođen je u Novom Sadu 1992. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Softversko inženjerstvo, odbranio je 2018.god. kontakt: kupcodanex@outlook.com