UDC: 004.42:004.43 Info M: str. 4-10

IMPLEMENTACIJA JEZIKA SPECIFIČNIH ZA DOMEN UPOTREBOM TEXTX ALATA
IMPLEMENTATION OF DOMAIN-SPECIFIC LANGUAGES USING THE TEXTX TOOL

Igor Dejanović, Renata Vaderna, Gordana Milosavljević, Željko Vuković
Fakultet tehničkih nauka, Univerzitet u Novom Sadu

REZIME: Jezici specifični za domen (Domain-Specific Languages - DSL) su jezici fokusirani i ograničeni na uzak domen ljudske delatnosti. Iako se koriste već više decenija poslednjih godina je primetan porast interesovanja za njih, što je, pre svega, uzrokovano dostupnošću alata za njihovu jednostavnu izradu i evoluciju. U ovom radu opisan je textX alat razvijen na Katedri za informatiku Fakulteta tehničkih nauka u Novom Sadu, i njegova upotreba za razvoj jezika specifičnih za domen. textX, polazeći od gramatike jezika, dinamički u vreme izvršavanja kreira meta-model i parser datog jezika. Parser je sposoban da iz tekstualnog opisa modela, odnosno programa, rekonstruiše memorijski model koji je usklađen sa datim meta-modelom. Implementiran je na programskom jeziku Python. textX je slobodan softver otvorenog koda dostupan na GitHub-u.

KLJUČNE REČI: jezici specifični za domen, parser, meta-jezik, meta-model, interpreter, generator programskog koda, alat.

ABSTRACT: Domain-Specific Languages (DSLs) are languages constrained and focused on a particular domain of human endeavor. Although they have been used for several decades, in the last years it is evident an increased interest in their usage which is caused by an increased availability of tools for their easy production and evolution. In this paper we describe textX tool developed at Chair for informatics, Faculty of technical sciences in Novi Sad and its usage in development of DSLs. Starting from the grammar textX creates meta-model and parser of the given language in run-time. Parser is capable to, out of the textual model description, produce memory model which conforms to language meta-model. textX is implemented on Python programming language as a free and open-source project available at GitHub.

KEY WORDS: Domain-Specific Languages, parser, meta-language, meta-model, interpreter, source code generator, tool.

UVOD

Upotreba apstrakcija prilagođenih domenu omogućava jezicima specifičnih za domen (*Domain-Specific Languages - DSL*) da postignu visok nivo ekspresivnosti i podignu produktivnost i do 1000% [1, 2]. U poređenju sa jezicima opšte namene (*General-Purpose Languages - GPL*), DSL-ovi su lakši za razumevanje [3, 4]. Takođe, DSL-ovi su jednostavniji za analizu, verifikaciju, optimizaciju, paralelizaciju i transformaciju specifičnu za domen [5].

Ipak, i pored njihove superiornosti veliku prepreku u prihvatanju imala je cena razvoja koja je u prošlosti bila visoka. Kompajlere i alate za novi jezik je bilo teško praviti i održavati. Poslednjih godina sve su popularniji alati koji imaju za cilj da značajno pojednostave proces razvoja i samim tim redukuju njegovu cenu. Ovi alati se jednim imenom nazivaju *jezičke radionice* (*Language Workbenches*) [6].

Iako jezičke radionice nude veliku pomoć u kreiranju jezičke infrastrukture i dalje su često vezane za određena integrisana okruženja što otežava njihovu upotrebu u različitim scenarijima. Takođe, proces izrade jezika je složen u smislu da svaka izmena jezika i testiranje podrazumeva generisanje čitave infrastrukture i pokretanje nove instance integrisanog okruženja. Tzv. razvojni okret (*round-trip*) je nedopostivo dug. Ovakav način rada, gde se često očekuje od programera da sačeka da se kreiraju potrebne komponete, demotiviše izradu jezika kroz eksperimentiranje.

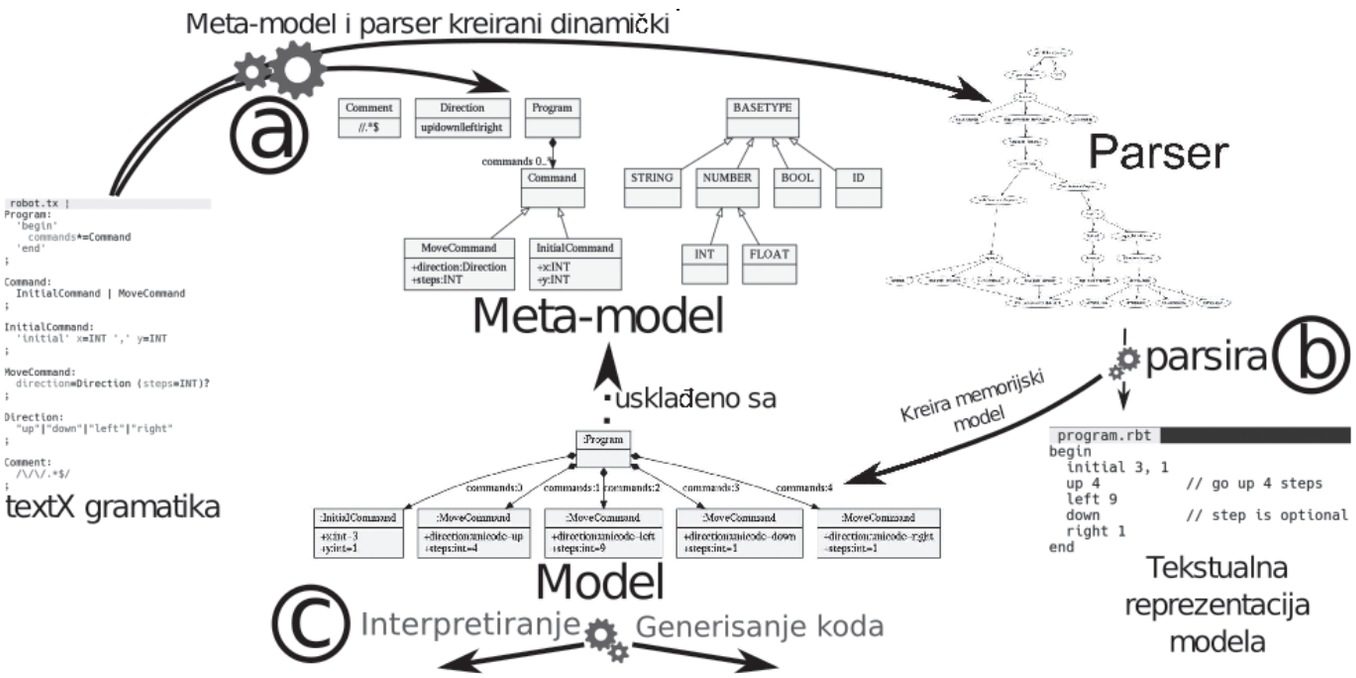
textX [7] alat je nastao sa ciljem da omogući brz razvojni okret. To je omogućeno interpreterskim radom alata. Na osnovu gramatike textX se konfiguriše za prepoznavanje iskaza na novom jeziku. Implementiran je na bazi Arpeggio parsera [8, 9] koji je takođe razvijen na našoj katedri. Upotrebom `textX` i `Arpeggio` alata razvijeno je više jezika kako u domenu slobodnog softvera [10, 11, 12], tako i u domenu industrijskog softvera [13].

Na slici 1 prikazana je arhitektura alata i proces rada. Na osnovu gramatike jezika textX dinamički kreira meta-model jezika i parser (a). Parser će, pri parsiranju modela/programa na novom DSL-u (b), dinamički kreirati memorijski model na bazi Python objekata koji će odgovarati meta-modelu. Model se dalje može interpretirati ili prevoditi u druge izvršive oblike kroz, na primer, generisanje programskog koda (c) za neku ciljnu platformu.

textX je slobodan projekat otvorenog koda dostupan na GitHub-u. Distribuira se pod MIT licencom što ga čini pogodnim kako za razvoj projekata otvorenog koda tako i za upotrebu na komercijalnim projektima.

Idejno najbliži našem alatu je Xtext alat [14] po čijem uzoru je textX i nastao. Xtext je baziran na programskom jeziku Java i istoj ideji kreiranja meta-modela i parsera iz jednog opisa (gramatike) ali je način rada fundamentalno drugačiji. Xtext radi kao kompajler/generator jer se i parser i sva ostala jezička infrastruktura generiše. Ovo ima za posledicu sporiji razvoni okret jer je pri svakoj promeni gramatike potrebno regenerisati kod parsera i jezičkih priključaka za Eclipse integrisano okruženje i pokrenuti novu kopiju razvojnog okruženja. Ovakav način rada usporava programera i otežava razvoj kroz eksperimentisanje. Xtext se razvija već dugi niz godina, ima veliku zajednicu i iako je projekat otvorenog koda iza njega stoji nemačka firma Itemis čiji programeri su inicijalni autori projekta i dalje aktivno rade na njegovom razvoju.

Sličan projekat je Spoofax [15] koji je takođe baziran na programkom jeziku Java. Iako nema industrijsku snagu i prime-



Slika 1. *textX pristup*

nu kao Xtext vredi ga spomenuti jer je razvoji okret kod njega brži što je postignuto posebnom tehnikom dinamičkog učitavanja jezičke infrastrukture. Ova tehnika obezbeđuje da se u istoj instanci integrisanog okruženja jezik može i razvijati i testirati.

U okviru Python zajednice nije nam poznat alat sličan textX-u. Jezici i parseri se uglavnom razvijaju upotrebom klasičnih parser generatora ili interpretera sličnih Arpeggio parseru. Neki od značajnijih predstavnika u ovom domenu su pyparsing [16], PLY [17] i parsimonious [18].

U nastavku rada biće opisan textX alat i dati primeri razvoja jezika specifičnih za domen. Struktura rada je sledeća: u drugoj sekciji opisana je gramatika meta-jezika (jezika za opis jezika); sekcija 3 opisuje pokretanje eksternog `textx` alata za proveru i vizualizaciju meta-modela i modela i daje pregled upotrebe alata kroz API; sekcija 4 daje primer pokaznog mini robot jezika dok sekcija 5 opisuje složeniji jezik za opis psihofizioloških eksperimenata za reakciono vreme; zaključak rada dat je u sekciji 6.

textX gramatika

textX gramatika se sastoji od skupa pravila. Svako pravilo ima jedinstveno ime definisano na početku, pre dvotačke, i telo pravila koje opisuje obrazac koji mora biti prepoznat od strane parsera, slično produkcijama kod EBNF. Pravilo se završava karakterom ';'. Pored obrasca za parsiranje, pravila, u isto vreme, definišu koncepte ciljnog jezika tj. njegove apstraktne sintakse, odnosno meta-modela. Ovi koncepti će u vreme izvršavanja biti dostupni kao *Python* klase i biće korišćeni za instanciranje objekata koje parser prepozna u ulaznom tekstualnom fajlu modela/programa.

Na primer, zamislimo da razvijamo jezik za opis crteža. Koncepti ovog jezika bi mogli biti `Shape`, `Line`, `Circle` itd. Sledeće pravilo se koristi da opiše koncept `Circle`:

```
Circle:
  'Circle' '(' color=ID ',' size=INT ')'
;
```

Prva linija definiše ime pravila. Druga linija sadrži telo pravila. Pravilo se završava na trećoj liniji karakterom ';'. U ovom pravilu vidimo dva osnovna gradivna bloka. Tekst pod znacima navoda (npr. 'Circle') se u textX-u naziva *string match* i predstavlja instrukciju textX-u da u ulaznom stringu na tekućoj lokaciji očekuje identičan string. Dakle, svaki krug mora početi sa ključnom rečju `Circle` i otvorenom zagradom. String koji se prepozna sa ovakvom konstrukcijom se u toku kreiranja objektnog grafa odbacuje jer se smatra delom konkretne sintakse isključivo potrebne parseru (i programeru) da prepozna objekat, a za apstraktnu sintaksu nije potreban.

Drugi gradivni element se naziva *dodela* (eng. *assignment*), na primer `color=ID`. Dodela daje instrukciju šta treba da se parsira, na desnoj strani (eng. *right hand side* - RHS) i naziv atributa koji će biti dinamički kreiran na instanci klase opisane pravilom na svojoj levoj strani (eng. *left hand side* - LHS). U primeru `Circle` biće kreirana dva atributa za *Python* klasu `Circle`: `color` tipa `string` i `size` tipa `integer`.

Za svako pravilo dato gramatikom textX će u fazi instanciranja meta-modela kreirati *Python* klasu. Atributi instanci svake klase su unapred poznati na osnovu pravila dodele. Parser pridružen meta-modelu će instancirati odgovarajuću klasu kada naiđe u ulazu na odgovarajući obrazac.

```
1  EntityModel:
2    entities+=Entity
3  ;
4
5  Entity:
6    'entity' name=ID '{'
7      attributes*=Attribute
8    '}'
9  ;
10
11 Attribute:
12   name=ID ':' type=[Entity]
13 ;
14
15 Comment:
16   /\/\/.*$/
17 ;
```

a)

```
1  entity Person {
2    name : string
3    address: Address
4    age: integer
5  }
6
7  entity Address {
8    street : string
9    city : string
10   country : string
11 }
```

b)

Slika 2. *textX bazirani a) meta-model i b) primer modela jezika Entity.*

Na slici 2 prikazan je jednostavan jezik za modelovanje podataka sličan sa jezikom opisanim u [19]. Na levoj strani slike (a) data je gramatika jezika dok je na desnoj strani (b) dat model napisan na datom jeziku.

Na ovom primeru vidimo nekoliko novih gradivnih blokova. Prvo pravilo u gramatici ima posebnu ulogu jer će biti korensko pravilo parsera kao i kreiranih modela (pravilo `EntityModel` na liniji 1 na slici 2-a). Parsiranje započinje prepoznavanjem korenskog pravila i na kraju parsiranja biće vraćena instanca klase `EntityModel`.

Postoji još jedno specijalno pravilo. To je pravilo `Comment` (linije 15-17, slika 2-a). Ovo pravilo definiše komentare jezika. Sve što može biti prepoznato ovim pravilom smatra se komentarom. textX će pokušati da prepozna ovo pravilo između svaka druga dva prepoznavanja i, ukoliko uspe, deo ulaza koji je prepoznat kao komentar biće odbačen iz dalje analize. Zapravo, komentari se čuvaju ali se trenutno ne koriste. U budućim verzijama moguće je da će komentari, kao i prazni karakteri (*whitespaces*) biti korišćeni u npr. generisanju koda iz modela.

Na liniji 2, uz pomoć dodele +=, definišemo da će svaki model da sadrži jednu ili više `Entity` instanci. Pored dodele tipa *jedan ili više* podržana je i dodela *nula ili više* (`*=`, linija 7) kao i *opciona dodela* (`?=`). Prva dva pravila će uvek rezultovati u atributu objekta čiji tip će biti *Python* lista. Kod opcione dodele atribut će biti *boolean* tipa a vrednost će biti istinita (`True`) ako je prepoznavanje, definisano desnom stranom dodele, uspešno, odnosno biće laž (`False`) ukoliko nije.

Navođenje imena nekog drugog pravila unutar tela pravila (na primer `Entity` na liniji 2) naziva se *referenciranje pravila* (eng. *rule reference*). Referenciranje pravila definiše relacije između koncepata jezika. Postoje dva tipa referenciranja pravila: referenciranje preko prepoznavanja (eng. *match rule reference*) i referenciranje preko veze (eng. *link rule reference*). Semantika referenciranja preko prepoznavanja je da se na mestu reference očekuje prepoznavanje nove instance ciljnog pravila. Instanca referenciranog pravila biće sadržana u kontekstu objekat pravila koje referencira. Ovaj tip referenciranja obavlja se prostim navođenjem imena ciljnog pravila, najčešće u RHS. Kod referenciranja preko veze, parser očekuje ime ciljnog objekta dok sam objekat mora biti definisan negde drugo u fajlu. textX će izvršiti automatsko razrešavanje reference i LHS atribut će biti Python referenca na ciljni objekat. Primer referenciranja preko veze je dat na liniji 12 slike 2-a (veza `type` atributa na `Entity` instancu).

Na linijama 5-9 u gramatici na slici 2-a definisan je `Entity` koncept. Možemo videti da svaki entitet počinje sa ključnom rečju `entity`, da ima ime (`name`) i listu atributa (`attributes`). Svaki entitet mora imati nula ili više atributa (`*=`) unutar tela entiteta. Attributi su sadržani unutar entiteta.

Na linijama 11-13 gramatike data je definicija atributa. `Attribute` je koncept koji ima ime (`name`) tipa `ID` i tip (`type`) koji je veza za entitet (pravilo `Entity`). Znači, na ovom mestu u programu mora se naći ime postojećeg entiteta.

VALIDACIJA, VIZUELIZACIJA I INSTANCIRANJE META-MODELA I MODELA

Meta-model definisan gramatikom je moguće validirati i vizuelizovati na dva načina:1) upotrebom `textx` alata, 2) kroz `textX` API.

`textx` alat se poziva sa komandne linije i kao parametre prima komandu `check` ili `visualize`, putanju do gramatike tj. meta-modela i opciono putanju do modela.

Na primer:

```
$ textx visualize robot.tx program.rbt
Meta-model OK.
Model OK.
Ili ukoliko želimo da vizualizujemo (meta)model:
$ textx visualize robot.tx program.rbt
Meta-model OK.
Model OK.
Generating 'robot.tx.dot' file for meta-model.
To convert to png run 'dot -Tpng -O robot.tx.dot'
Generating 'program.rbt.dot' file for model.
To convert to png run 'dot -Tpng -O program.rbt.dot'
```

Prethodna komanda će kreirati tekstualne `dot` fajlove. `dot` je DSL za opis grafova i deo je GraphViz paketa [20]. Ovi fajlovi se mogu videti specijalizovanim pregledačima ili se transformisati u vektorske ili bitmapirane slike. Na slici 5 možemo da vidimo kako izgleda meta-model i model za robot DSL. Slika je generisana upotrebom prethodne komande.

Drugi način da proverimo i vizuelizujemo meta-model je kroz `textX` API. Ovo je ujedno i način kako ćemo koristiti naš novi jezik u praksi. Da bi instancirali meta-model i parsirali naše modele iz programskog koda u Python-u postupamo kao na slici 3.

```
from textx.metamodel import metamodel_from_file
from textx.export import metamodel_export, model_export

# Kreiramo Entity meta-model
entity_mm = metamodel_from_file('entity.tx')

# Parsiramo model. person_model će biti instanca EntityModel
person_model = entity_mm.model_from_file('person.ent')

# Opciono, vizualizujemo/eksportujemo metamodel i model u dot
metamodel_export(entity_mm, 'entity_metamodel.dot')
model_export(person_model, 'person_model.dot')
```

Slika 3. *Upotreba textX biblioteke iz Python programskog koda.*

`person_model` će biti instanca `EntityModel` pa će imati atribut `enitities` koji će biti lista `Entity` objekata. Dalje, koristimo ovaj objektni graf kao i sve druge objekte. Možemo vršiti interpretiranje modela ili generisanje programskog koda. Primeri interpretacije i generisanja programskog koda mogu se pronaći u repozitorijumu projekta.

PRIMER - ROBOT JEZIK

U ovoj sekciji opisan je mini jezik za pomeranje hipotetičkog robota po zamišljenoj tabli. Ovaj jezik služi kao pokazna vežba da prikaže osnove upotrebe textX alata.

Robot jezik definiše niz instrukcija za pomeranje robota kao i instrukciju za postavljanje robota na početnu poziciju. Robot se kreće u četiri smera: gore (*up*), dole (*down*), levo (*left*) i desno (*right*). Pri svakoj instrukciji pomeranja možemo opciono navesti broj koraka. Ukoliko se broj koraka ne navede smatra se da je 1. Tabla ima numerisane redove i kolone. I red i kolona (koordinate x i y) su celi brojevi.

Validan primer programa na ovom jeziku je prikazan na slici 4-b. Ključne reči `begin` i `end` predstavljaju početak i kraj programa. Komanda `initial` služi za postavljanje robota na početnu poziciju i prima koordinate x i y koji su celobrojnog tipa.

Gramatika robot jezika data je na slici 4-a. Pravilo `Program` definisano na linijama 1-5 definiše strukturu programa. Svaki program počinje ključnom rečju 'begin', sastoji se od nula ili više komandi (linija 3) i završava se ključnom rečju 'end'.

Komanda može biti komanda za pomeranje (`MoveCommand`) ili komanda za postavljanje na početnu poziciju (`InitialCommand`), što je definisano linijama 7-9.

Komanda za postavljanje robota na početnu poziciju opisana je na linijama 11-13. Komanda mora početi ključnom rečju `initial` iza koje se navode koordinate x i y razdvojene zarezom. Koordinate su celobrojnog tipa, što je podržano textX ugrađenim `INT` pravilom.

Komanda za pomeranja ima pravac (atribut `direction`) i opcioni broj koraka (atribut `steps`) koji je takođe celobrojnog `INT` tipa.

Pravac je definisan na linijama 19-21 i predstavlja uređeni izbor jedne od ključnih reči: `up`, `down`, `left`, `right`.

Ovako definisana gramatika određuje meta-model koji može da se predstavi grafički u formi dijagrama sličnog UML dijagramu klasa koji se može automatski generisati kao što je opisano u prethodnoj sekciji. Na slici 5-a je dat meta-model robot jezika. Sa dijagrama jasno vidimo da se svaki robot program sastoji od nula ili više komandi koje mogu biti `InitialCommand` ili `MoveCommand`. Slika 5-b daje grafički prikaz objektnog grafa za program sa slike 4-b.

Kompletan primer robot jezika sa opisom interpretera je dat u textX dokumentaciji.

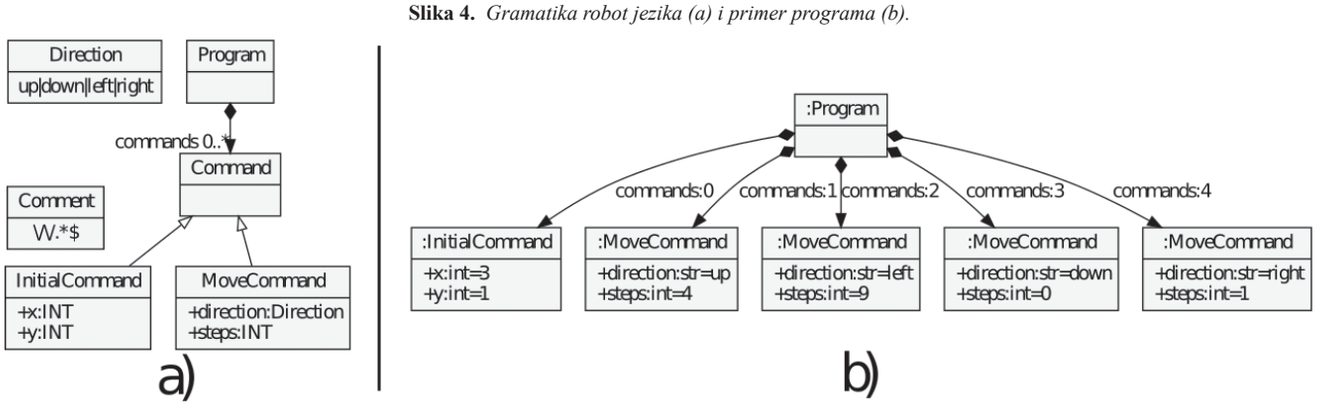
```
1 Program:
2   'begin'
3     commands*=Command
4   'end'
5 ;
6
7 Command:
8   InitialCommand | MoveCommand
9 ;
10
11 InitialCommand:
12   'initial' x=INT ',' y=INT
13 ;
14
15 MoveCommand:
16   direction=Direction (steps=INT)?
17 ;
18
19 Direction:
20   "up"|"down"|"left"|"right"
21 ;
```

a)

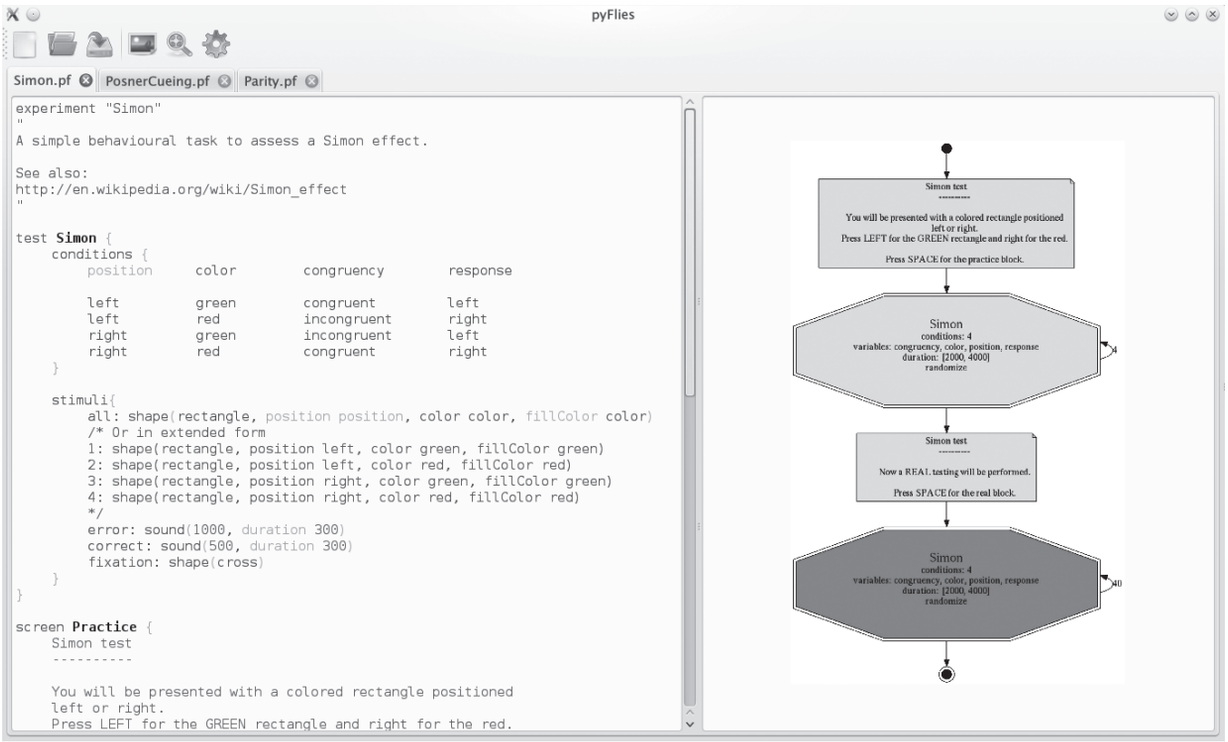
```
begin
  initial 3, 1
  up 4          // idi gore 4 koraka
  left 9
  down          // korak je opcion
  right 1
end
```

b)

Slika 4. *Gramatika robot jezika (a) i primer programa (b).*



Slika 5. Meta-model (a) i primer modela (b) robot DSL-a.



Slika 6. *pyFlies editor i primer eksperimenta.*

PRIMER - PYFLIES JEZIK

pyFlies jezik [11] je jezik za modelovanje psihofizioloških eksperimenata za merenje reakcionog vremena. Jezik i integrisano okruženje za razvoj predstavljaju slobodan softver otvorenog koda. pyFlies je primer kreiranja složenog DSL-a upotrebom textX alata.

Na slici 6 prikazano je okruženje za razvoj. Na levoj strani definiše se model eksperimenta upotrebom DSL-a. Na desnoj strani vidimo automatski generisanu vizualizaciju strukture eksperimenta.

Eksperiment opisan pyFlies DSL-om se automatizovano prevodi na ciljne platforme putem generisanja programskog koda. Trenutno je potpuno podržana PsychoPy [21] biblioteka dok su podrška za Expyriment [22] i jsPsych [23] u razvoju.

Jedan od osnovnih dizajn principa pri razvoju ovog jezika je da bude što bliži načinu rada eksperimentalnih psihologa pri modelovanju testa i eksperimenta. Test reakcionog vremena izvodi se u jednoj ili više serija gde serija ima više ponavljanja. Prilikom testa zahteva se od subjekta da na prezentovani stimulus što brže da adekvatan odgovor. Najčešće je odgovor pritisak određenog tastera, stiskanje ručice i sl.

Pri modelovanju testa definišu se varijable čije kombinacije vrednosti određuju uslove pri svakom ponavljanju. Kombinacije vrednosti ovih varijabli, često se čuvaju u nekom tabelarnom formatu (npr. LibreOffice Calc, ili MS Excel). Stoga je bilo važno da se u novom DSL-u simulira tabelarni način rada.

Na slici 7-b prikazan je opis tzv. Sajmonovog testa na pyFlies jeziku. Tablica uslova definisana je u sekciji `conditi-`

```
TestType:
  "test" name=ID "{"
    'conditions' '{'
      /\s*/
      condVarNames+=WORD[eolterm]
      conditions+=Condition
    '}'

    'stimuli' '{'
        condStimuli+=ConditionStimuli
        ('duration' duration=Duration)?
    '}'
  "}"
;

Condition:
  /\s*/
  varValues+=WORD[eolterm]
;

WORD:
  INT|/[-\w]*\b/
;
```

a)

```
test Simon {
    conditions {
        position     color        congruency        response

        left         green        congruent         left
        left         red          incongruent       right
        right        green        incongruent       left
        right        red          congruent         right
    }

    stimuli{
        all: shape(rectangle, position position, color color
                   fillColor color)
        error: sound(1000, duration 300)
        correct: sound(500, duration 300)
        fixation: shape(cross)
    }
}
```

b)

Slika 7. *Deo pyFlies gramatike (a) i primer testa (b).*

`ons` na linijama 2-9. Vidimo da za ovaj test imamo 4 varijable (`position`, `color`, `congruency` i `response`).

Tablica uslova opisana je delom gramatike na linijama 3-7 na slici 7-a. Zaglavlje tablice uslova čine nazivi varijabli. Ovaj deo ulaznog modela se parsira sa delom gramatike na liniji 5. Vidimo da će atribut `condVarNames` da bude Python lista koja će sadržati jedan ili više (+= dodela) stringova prepoznatih od strane pravila `WORD`. Ukoliko bi koristili standardno prepoznavanje ovo pravilo bi osim zaglavlja parsiralo i ostatak tablice pošto je separator između zaglavlja i vrednosti varijabli znak za novi red. textX omogućava da se na elegantan način definišu ovakva specijalna pravila upotrebom tzv. modifikator prepoznavanja ponavljanja (eng. *repetition modifier*). U ovom slučaju to je modifikator `eolterm` koji će ograničiti prepoznavanje unutar jednog reda. Dakle, upotrebom ovog modifikatora atribut `condVarNames` će sadržati samo nazive varijabli iz prvog reda tablice, odnosno zaglavlja. Na isti način je definisano i prepoznavanje pojedinačnog stanja. Vrednosti varijabli za jedno stanje moraju biti navedeni unutar jednog reda (linija 18).

Iz ovako opisanog eksperimenta generiše se programski kod na zadatoj ciljnoj platformi. Konkretno, ukoliko je ciljna platforma PsychoPy generiše se Python kod. Prednost upotrebe pyFlies jezika je da model mogu da kreiraju i ne-programeri jer se radi sa konceptima domena eksperimentalne psihologije. Za direktnu upotrebu PsychoPy-a potrebno je programersko znanje. I pored potrebnog znanja, odnos broja linija koda potrebnih za opis eksperimenta na pyFlies jeziku u odnosu na direktnu upotrebu PsychoPy plaforme ide i do 10 puta u korist pyFlies jezika.

Detaljniji opis jezika kao i niz primera mogu se pronaći u repozitorijumu pyFlies projekta.

ZAKLJUČAK

U ovom radu opisana je implementacija jezika specifičnih za domen uz pomoć textX alata razvijenog na Katedri za informatiku Fakulteta tehničkih nauka. textX se sa uspehom koristi kako u akademiji tako i u industriji. textX je slobodan projekat otvorenog koda i koristi MIT licencu tako da ga je moguće koristiti kako u drugim projektima otvorenog koda tako i u komercijalnim projektima.

Jedan od osnovnih ciljeva pri izradi textX-a bio je da se omogući jednostavnost u razvoju uz brz razvojni okret što promoviše razvoj jezika kroz eksperimentisanje. Cilj je postignut izborom interpreterskog načina rada.

textX nije zavisan od određenog integrisanog okruženja tako da se može koristiti u različitim scenarijima i različitim projektima. Iz modela/programa pisanih na novom DSL-u može se generisati programski kod ili se model može direktno interpretirati.

Prikazali smo dva primera upotrebe textX-a. Robot jezik je jednostavan jezik koji služi kao pokazna vežba kako se textX koristi dok je pyFlies realni DSL koji se koristi za modelovanje psihofizioloških eksperimenata.

Dalji pravci razvoja biće orijentisani ka izradi infrastrukture za jednostavnije kreiranje generatora programskog koda kao i podrške za integraciju sa različitim editorima programskog koda.

REFERENCE

[1] Steven Kelly and Juha-Pekka Tolvanen. *Visual domain-specific modeling: Benefits and experiences of using metacase tools*. In International Workshop on Model Engineering, at ECOOP, volume 2000. Citeseer, 2000.

[2] Steven Kelly and Juha-Pekka Tolvanen. *Domain-Specific Modeling: Enabling Full Code Generation*. Wiley-IEEE Computer Society Pr, March 2008.

[3] T. Kosar, N. Oliveira, M. Mernik, M.J. Pereira, M. Crepinsek, D. Cruz, P. Henriques. *Comparing general-purpose and domain-specific languages: an empirical study*. Computer Science and Information Systems - ComSIS, 7(2):247–264, April 2010. Special Issue.

[4] Tomaž Kosar, Marjan Mernik, and Jeffrey C Carver. *Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments*. Empirical software engineering, 17(3):276–304, 2012.

[5] M. Mernik, J. Heering, and A.M. Sloane. *When and how to develop domain-specific languages*. ACM Computing Surveys (CSUR), 37(4):316–344, 2005.

[6] Sebastian Erdweg, Tijs van der Storm, Markus Völter, Laurence Tratt, Remi Bosman, William R Cook, Albert Gerritsen, Angelo Hulshout, Steven Kelly, Alex Loh, et al. *Evaluating and comparing language workbenches: Existing results and benchmarks for the future*. Computer Languages, Systems & Structures, 44:24–47, 2015.

[7] I. Dejanović, G. Milosavljević, and R. Vaderna. *Arpeggio: A flexible PEG parser for python*. Knowledge-Based Systems, 95:71 – 74, 2016.

[8] *textX – Alat za implementaciju jezika specifičnih za domen na programskom jeziku Python*, Online https://github.com/igordejanovic/textX, poslednji pristup 24.05.2016.

[9] Igor Dejanović, Gordana Milosavljević. *Performance Evaluation of the Arpeggio Parser*. In 4rd International Conference on Information Society Technology and Management (ICIST 2014), volume 1, pages 229–234, Kopaonik, Serbia, 2014.

[10] Miloš Simić, Željko Bal, Renata Vaderna, Igor Dejanović. *PyTabs: A DSL for simplified music notation*. In 5rd International Conference on Information Society Technology and Management (ICIST 2015), volume 1, pages 405–409, Kopaonik, Serbia, 2015. Information Society of Serbia. ISBN:978-86-85525-16-2.

[11] *pyFlies - DSL za opis experimenata reakcionog vremena*. Online https://github.com/igordejanovic/pyflies, 2016. poslednji pristup 24.05.2016.

[12] Milan Kosanović, Igor Dejanović, Gordana Milosavljević. *Applang – a dsl for specification of mobile applications for android platform based on textx*. In To be published, 2016.

[13] Alen Suljkanović, Gordana Milosavljević, Dušan Majstorović, Igor Dejanović. *Implementacija generičke dsp komponente u sklopu typhoon hil softvera za modelovanje šema energetskih sklopova*. Info M – Časopis za informacione tehnologije i multimedijalne sisteme, (54):43–49, 2015. ISSN 1451-4397.

[14] Moritz Eysholdt, Heiko Behrens. *Xtext: implement your language faster than the quick and dirty way*. Proceedings of the

ACM international conference companion on Object oriented programming systems languages and applications companion, pages 307-309, 2010.
[15] Kats Lennart CL, Eelco Visser. *The spoofax language workbench: rules for declarative specification of languages and IDEs*. Vol. 45. No. 10. ACM, 2010.
[16] McGuire, Paul. *Getting started with pyparsing*. O'Reilly Media, Inc., 2007.
[17] PLY- Python Lex-Yacc, Online http://www.dabeaz.com/ply/. Poslednji pristup 24.05.2016.
[18] Parsimonious, Online https://github.com/erikrose/parsimonious. Poslednji pristup 24.05.2016.
[19] Igor Dejanović, Gordana Milosavljević, Branko Perišić, Maja Tumbas. *A Domain-Specific Language for Defining Static Structure of Database Applications*. Computer Science and Information Systems, 7(3):409–440, June 2010.
[20] John Ellson, Emden Gansner, Lefteris Koutsofios, Stephen North, Gordon Woodhull, Short Description, Lucent Technologies. *Graphviz — open source graph drawing tools*. In Lecture Notes in Computer Science, pages 483–484. Springer-Verlag, 2001.
[21] Jonathan W Peirce. *PsychoPy—psychophysics software in Python*. Journal of neuroscience methods, 162(1):8–13, 2007.
[22] Florian Krause, Oliver Lindemann. Expyriment: *A Python library for cognitive and neuroscientific experiments*. Behavior research methods, 46(2):416–428, 2014.
[23] Joshua R de Leeuw. *jsPsych: A JavaScript library for creating behavioral experiments in a Web browser*. Behavior research methods, pages 1–12, 2014. https://github.com/jodeleeuw/jsPsych.

Dr Igor Dejanović, Univerzitet u Novom Sadu, Fakultet tehničkih nauka, Katedra za informatiku
Kontakt: igord@uns.ac.rs
Oblasti interesovanja: Jezici specifični za domen, Modelima upravljan razvoj softvera, Upravljanje konfiguracijom softvera

MSc Renata Vaderna, Univerzitet u Novom Sadu, Fakultet tehničkih nauka, Katedra za informatiku
Kontakt: vrenata@uns.ac.rs
Oblasti interesovanja: Teorija grafova, Crtanje grafova, Jezici specifični za domen, Modelima upravljan razvoj softvera

Dr Gordana Milosavljević, Univerzitet u Novom Sadu, Fakultet tehničkih nauka, Katedra za informatiku
Kontakt: grist@uns.ac.rs
Oblasti interesovanja: Modelima upravljan razvoj softvera, Agilne metodologije, Razvoj informacionih sistema vođen modelima

MSc Željko Vuković, Univerzitet u Novom Sadu, Fakultet tehničkih nauka, Katedra za informatiku
Kontakt: zeljkov@uns.ac.rs
Oblasti interesovanja: Modelima upravljan razvoj softvera, Integracija poslovnih aplikacija, Računarske mreže

UPUTSTVO ZA PRIPREMU RADA

1. Tekst pripremiti kao Word dokument, A4, u kodnom rasporedu 1250 latinica ili 1251 ćirilica, na srpskom jeziku, bez slika. Preporučen obim – oko 10 strana, single prored, font 11.
2. Naslov, abstakt (100-250 reči) i ključne reči (3-10) dati na srpskom i engleskom jeziku.
3. Jedino formatiranje teksta je normal, bold, italic i bolditalic, VELIKA i mala slova (tekst se naknadno prelama).
4. Mesta gde treba ubaciti slike, naglasiti u tekstu (Slika1...)
5. Slike pripremiti odvojeno, VAN teksta, imenovati ih kao u tekstu, radi identifikacije, u sledećim formatima: rasterske slike: jpg, tif, psd, u rezoluciji 300 dpi 1:1 (fotografije, ekranski prikazi i sl.), vektorske slike – cdr, ai, fh,eps (šeme i grafikoni).
6. Autor(i) treba da obavezno priloži svoju fotografiju (jpg oko 50 Kb), navede instituciju u kojoj radi, kontakt i 2-4 oblasti kojima se bavi.
7. Maksimalni broj autora po jednom radu je 5.

Redakcija časopisa Info M