Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/04BE15Bal

RAZVOJ JEZIKA ZA OPIS KOMANDNIH KORISNIČKIH INTERFEJSA

DEVELOPMENT OF A LANGUAGE FOR SPECIFICATION OF COMMAND USER INTERFACES

Željko Bal, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Ovaj rad opisuje implementaciju jezika specifičnog za domen opisivanja korisničkih interfejsa, pomoću koga je moguće izgenerisati kod za interfejs komandne linije, kao i za grafički korisnički interfejs određene aplikacije.*

Ključne reči: *Jezik specifičan za domen, JSD, textX, CLI, GUI*

Abstract – *This thesis describes the implementation of a domain specific language for describing user interfaces, that can be used to generate the code for a command line interface, as well as a graphical user interface for a certain application.*

Keywords: *Domain specific language, DSL, textX, CLI, GUI*

1. UVOD

U ovom radu je opisan jezik specifičan za domen opisivanja komandnih korisničkih interfejsa nazvan „Command Interface Description Language" (u daljem tekstu skraćeno CID). Glavni pojmovi koji čine meta-model jezika su komande i njihovi parametri, čime se opisuje hijerarhijska struktura komandi određenog korisničkog interfejsa. Na osnovu ovog opisa CID alat može da generiše korisnički interfejs komandne linije, kao i grafički korisnički interfejs, koji se oslanja na izgenerisani ili već postojeći interfejs komandne linije. Na ovaj način se omogućava lako kreiranje grafičkog interfejsa i interfejsa komandne linije za određenu aplikaciju na osnovu jedinstvene specifikacije, što ubrzava razvoj i olakšava održavanje oba interfejsa. Takođe je moguće na lak način opisati već postojeći interfejs komandne linije neke aplikacije i izgenerisati grafički interfejs koji se oslanja na postojeći korisnički interfejs.

Gramatika CID jezika je opisana u TextX [1] meta-jeziku za opisivanje jezika specifičnih za domen. Na osnovu ovog opisa je generisan parser za sam CID jezik. Generatori za korisničke interfejse u okviru CID alata su implementirani u programskom jeziku Python, dok je za samo generisanje koda interfejsa korišćen šablon-generator Jinja 2 [2]. Korisnički interfejs komandne linije koji se generiše pomoću CID alata je implementiran u programskom jeziku Python, dok je grafički interfejs implementiran kao Electron [3] desktop aplikacija. Sam CID alat poseduje grafički interfejs i interfejs komandne linije koji su izgenerisani na osnovu opisa u CID jeziku.

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

2. TEORIJSKE OSNOVE

2.1 Jezici Specifični za Domen

Jezici specifični za domen (eng. *Domain Specific Language* - DSL) su računarski jezici koji su specijalizovani za određenu oblast primene (domen), za razliku od jezika opšte namene koji su široko primenjivi u različitim oblastima [4]. Jezici specifični za domen barataju pojmovima iz određenog usko definisanog domena i na taj način omogućavaju da se rešenja za određenu klasu problema iskažu jasnije i konciznije nego pomoću jezika opšte namene. Takođe, na isti način, omogućavaju stručnjacima iz date oblasti da lakše razumeju rešenje ili čak učestvuju u izradi samog rešenja.

2.2 Korisnički Interfejs

Korisnički interfejs je tačka interakcije između računara i ljudi (korisnika). Korisnički interfejsi podrazumevaju bilo koju vrstu interakcije (kao što su slika, zvuk, pozicija, pokret i druge...) prilikom koje se prenose informacije između korisnika i računarskog sistema [5]. Cilj korisničkih interfejsa je da omoguće lak, efikasan i prijatan način za upotrebu sistema koji proizvodi željeni rezultat [6]. U ovom radu će posebno biti razmatrana dva, trenutno najčešće korišćena, tipa korisničkih interfejsa na personalnim računarima: interfejs komandne linije (eng. *Command Line Interface* - CLI) i grafički korisnički interfejs (eng. *Graphical User Interface* - GUI).

2.3 Interfejs Komandne Linije

Interfejs komandne linije je način interakcije sa računarskim programom pri kom korisnik zadaje komande programu u vidu sukcesivnih redova teksta (komandnih linija) [7]. Interfejsi komandne linije omogućuju brže i konciznije zadavanje komandi od grafičkih korisničkih interfejsa i takođe omogućuju lakšu automatizaciju zadavanja komandi, zbog čega ih često preferiraju stručnjaci u oblasti računarstva.

2.4 Grafički Korisnički Interfejs

Grafički korisnički interfejs omogućuje korisniku interakciju sa računarskim programom putem "direktne manipulacije" grafičkim elementima [8]. Prosečni korisnici računara često vide grafički korisnički interfejs kao lakši za učenje i upotrebu od interfejsa komandne linije zbog vizuelnih metafora koje omogućavaju korisniku da poveže elemente interfejsa sa svojim mentalnim modelima. Iz istog razloga korisnici koji su navikli na upotrebu grafičkih korisničkih interfejsa često oklevaju da koriste aplikacije koje poseduju isključivo interfejs komandne linije.

3. KORIŠĆENE TEHNOLOGIJE

3.1 TextX

TextX [1] je meta jezik (jezik za opisivanje jezika) za specifikaciju jezika specifičnih za domen, napisan u programskom jeziku Python. Pomoću ovog meta jezika je moguće na lak način opisati gramatiku određenog tekstualnog jezika. Opis gramatike se sastoji od niza produkcionih pravila koja ujedno opisuju koncepte (apstraktnu sintaksu) i konkretnu sintaksu jezika. Na osnovu gramatike opisane na ovaj način TextX alat dinamički gradi meta-model i parser za opisani jezik.

Rezultat parsiranja dobijenim parserom je objektni model koji odgovara meta-modelu jezika i predstavlja apstraktno sintaksno stablo (graf) parsiranog teksta. Ovaj model je sačinjen od Python objekata, sa dinamički kreiranim klasama koje predstavljaju koncepte jezika opisane gramatikom. TextX projekat je inspirisan Xtext [9] alatom za izgradnju jezika specifičnih za domen, implementiranom u programskom jeziku Java i baziranom na Eclipse platformi [10].

TextX se u ovom projektu koristi za opisivanje gramatike samog CID jezika i parsiranje specifikacije korisničkih interfejsa, kao i za inicijalno pre-procesiranje dobijenog modela. Na ovaj način se smanjuje kompleksnost celog sistema korišćenjem gotovog rešenja za generisanje parsera, a ujedno se dobija i formalna definicija gramatike CID jezika.

3.2 Jinja

Jinja [2] je šablon-generator (eng. *template engine*) implementiran u programskom jeziku Python. Ovaj alat omogućava definisanje tekstualnih šablona i generisanje tekstualnog sadržaja na osnovu definisanih šablona i određenih ulaznih podataka. Definicija šablona sadrži statičku strukturu teksta koji se generiše, zajedno sa rezervisanim poljima na određenim mestima u tekstu, koje predstavljaju dinamički deo šablona i koje šablon-generator popunjava sa odgovarajućim prosleđenim ulaznim podacima prilikom generisanja.

U ovom projektu, Jinja se koristi za generisanje koda korisničkih interfejsa. Definisani su šabloni za Python objektni model koji koristi apstraktni parser argumenata komandne linije, kao i za HTML kôd i JavaScript objektni model koji se koristi za grafički korisnički interfejs.

3.3 Electron

Electron [3] je platformski nezavisan (eng. *cross platform*) programski okvir koji omogućava izgradnju desktop aplikacija korišćenjem *web* tehnologija (HTML, CSS i JavaScript). Electron se oslanja na Node.js okvir za *back-end*, pomoću koga se izvršava poslovna logika i komunikacija sa operativnim sistemom, implementirana u JavaScript jeziku.

Za *front-end*, Elecrton se oslanja na Chromium projekat [11] uz pomoć koga prikazuje grafički interfejs implementiran pomoću web tehnologija u desktop okruženju. Na ovaj način se postiže velika fleksibilnost pri kreiranju grafičkog interfejsa i mogućnost upotrebe širokog spektra *web* tehnologija, dok sa druge strane prikazani grafički interfejs ima isti izgled na svim podržanim platformama (Windows, Linux i OSX).

Electron je u ovom projektu korišćen za implementaciju generisanog grafičkog korisničkog interfejsa.

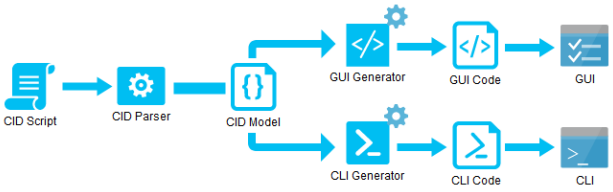
4. SPECIFIKACIJA I IMPLEMENTACIJA

4.1 Ciljevi

Glavni cilj sistema opisanog u ovom radu je da omogući lak i brz razvoj korisničkih interfejsa za aplikacije koje bi mogle imati koristi od dobrih osobina grafičkih korisničkih interfejsa i interfejsa komandne linije istovremeno. S obzirom na to da razvoj oba ova tipa korisničkih interfejsa za jednu aplikaciju uglavnom zahteva previše resursa, najčešće aplikacije poseduju jedan od ova dva tipa interfejsa, zavisno od toga koji od njih nudi benefite koji su bitniji za samu aplikaciju. Kako bi se prevazišao ovaj problem, u okviru ovog rada je razvijen sistem pomoću koga se mogu generisati ujedno interfejs komandne linije i grafički korisnički interfejs za određenu aplikaciju na osnovu jedinstvene specifikacije. Predviđena su dva slučaja korišćenja. U prvom je potrebno generisati grafički korisnički interfejs i interfejs komandne linije za novoimplementiranu aplikaciju (ili pak aplikaciju koja tek treba da bude implementirana). Ova dva izgenerisana korisnička interfejsa su kompatibilni jedan sa drugim u smislu da se grafički interfejs oslanja na interfejs komandne linije prilikom izvršavanje komande. U drugom slučaju korišćenja, već postoji implementiran interfejs komandne linije, za koji je potrebno generisati grafički interfejs. U ovom slučaju se opisuje već postojeći interfejs komandne linije i generiše se samo grafički interfejs koji se oslanja na postojeći interfejs aplikacije pri izvršavanju.

4.2 Arhitektura

CID jezik je implementiran kao eksterni jezik specifičan za domen. Osnovna komponenta sistema je parser CID jezika koji na osnovu definisane gramatike parsira specifikaciju koja opisuje korisnički interfejs. Parser vrši osnovnu validaciju specifikacije i kao rezultat vraća model interfejsa kao objektnu strukturu. Ovaj model se potom prosleđuje generatoru interfejsa komandne linije i/ili generatoru grafičkog korisničkog interfejsa. Generatori vrše dodatne validacije i transformacije modela specifične za tip korisničkog interfejsa i potom na osnovu modela generišu kôd za odgovarajuće korisničke interfejse. Na slici 1 je šematski prikazana arhitektura sistema:



Slika 1. *Šematski prikaz arhitekture CID sistema*

4.3 CID Parser

CID Parser kao ulaz ima jednu ili više datoteka koje sadrže opis korisničkog interfejsa u CID jeziku, dok kao rezultat vraća objektni model datog interfejsa, koji dalje može biti korišćen kao ulaz generatorima koda samih korisničkih interfejsa. *CID Parser* se, za samo parsiranje CID jezika i formiranje stabla apstraktne sintakse, oslanja na dinamički generisan parser na osnovu definisane gramatike jezika, pomoću TextX alata opisanog u odeljku 3.1. Nakon toga se stablo apstraktne sintakse validira i transformiše u model koji na ulazu očekuju generatori koda. Takođe, uloga *CID Parser* komponente je i da razreši reference na parametre i komande definisane

unutar iste ili odvojene datoteke. Objekti koji predstavljaju reference na parametre ili komande u stablu apstraktne sintakse bivaju zamenjeni instancama koje opisuju referencirane elemente. Samim tim, *CID Parser* parsira i kombinuje sve referencirane datoteke i kreira objektni model sa strukturom grafa.

4.4 CLI Generator

CLI Generator modul predstavlja generator koda interfejsa komandne linije. Ulaz u *CLI Generator* je model korsničkog interfejsa dobijen iz *CID Parser*-a, dok je izlaz kôd interfejsa komandne linije i primer koda koji implementira logiku aplikacije, a koji se poziva od strane izgenerisanog koda. Pre samog generisanja, *CLI Generator* dodatno procesira i validira model primenjujući transformacije i validacije koje su specifične za interfejs komandne linije. Nakon toga generator kopira statički kôd, koji uključuje generički parser argumenata komandne linije, koji parsira argumente na osnovu modela interfejsa komandne linije. Nakon kopiranja statičkog koda, generator generiše objektni model korisničkog interfejsa za generički parser argumenata. Generisanje koda se vrši pomoću Jinja šablon-generatora opisanog u odeljku 3.2. Prilikom ponovnog generisanja, generator ne menja deo koda koji implementira logiku same aplikacije, tako da korisnik može slobodno menjati ručno pisani kôd.

4.5 Generički CLI Parser

Generički CLI parser predstavlja parser argumenata komandne linije, koji na osnovu modela interfejsa komandne linije prepoznaje kojim komandama i parametrima pripadaju parsirani argumenti i kao rezultat daje objektni model datih argumenata. U okviru modela generičkog parsera su opisane komande korisničkog interfejsa i njihovi parametri.

Pre samog parsiranja argumenata, proverava se prisutnost argumenata koji iniciraju prikaz ugrađene korisničke pomoći definisanih za svaku komandu u okviru CID modela. Ukoliko se pronađe jedan od ovih argumenata, prikazuje se definisana korisnička pomoć za odgovarajuću komandu (prikazana na slici 2) i ostali argumenti se ignorišu.

Parsiranje argumenata komandne linije funkcioniše tako što se prvo argumenti klasifikuju po komandama kojima pripadaju i dalje na prepoznate ne-pozicione i preostale pozicione parametre. Zatim se za klasifikovane argumente, na osnovu modela, prepoznaju šabloni korišćenja za svaku od komandi i pri tom se prepoznaju i pozicioni parametri. Uz to se takođe proverava: da li su zadovoljeni šabloni korišćenja, da li postoje ne prepoznati argumenti ili argumenti koji su prepoznati, ali ne pripadaju prepoznatom šablonu korišćenja, kao i da prisustvo potkomande odgovara prepoznatom šablonu korišćenja. Potom se za vrednosti parametara koje nisu prisutne dodaju podrazumevane vrednosti definisane u modelu samih parametara i nakon toga se vrednosti svih parametara konvertuju u odgovarajuće Python tipove, u skladu sa definisanim tipovima podataka. Na posletku se proveravaju definisana ograničenja parametara i komandi i kao rezultat se vraća objektni model prepoznatih vrednosti parametara.

```
λ cid_generator --help -a
Generate a graphical and/or command line interface from a CID script.

Usage:  cid_generator <CID_FILE> <ROOT_COMMAND> <DEST_PATH> <sub_command>

Parameters:
  <ROOT_COMMAND>  The name of the entry point command of the interface.
  <CID_FILE>      The CID script file that contains a description of the command
                  interface to generate. (*.cid)
  <DEST_PATH>     The destination directory where to generate the interface.
  -h, --help      Shows a shorter help message. For this help message type:
                  cid_generator -h -a. All detailed help parameters: -a, --all.

Sub Commands:
  generate_cli   Generate a command line parser based on the provided description.
  generate_gui   Generate a graphical user interface based on the provided
                 description.
  generate_both  Generate both GUI and CLI.

For help about a speciffic sub command type: cid_generator <sub_command> -h.

CID (Command Interface Description) is a language for describing command interfaces (such as this one), that can be used
 to generate graphical or command line interfaces. For more information see https://github.com/zeljko-bal/CID.
```

Slika 2: *Prikaz korisničke pomoći za cid_generator komandu*

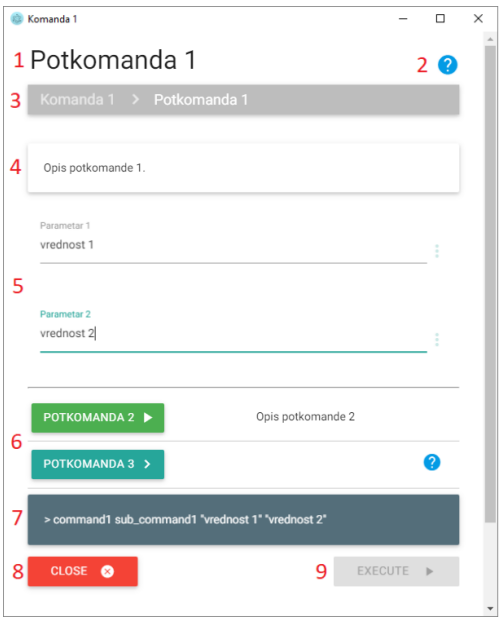
4.6 GUI Generator

GUI Generator modul predstavlja generator koda grafičkog korisničkog interfejsa. Proces generisanja koda je sličan gore opisanom procesu generisanja koda za interfejs komandne linije. Ulaz u *GUI Generator* je model korsničkog interfejsa dobijen iz *CID Parser*-a, dok je izlaz kôd grafičkog korisničkog interfejsa, koji se oslanja na interfejs komandne linije opisanog komandnog interfejsa. Pre samog generisanja, *GUI Generator* dodatno procesira i validira model primenjujući transformacije i validacije koje su specifične za grafički korisnički interfejs. Nakon toga generator kopira statički kôd, koji uključuje delove aplikacije grafičkog korisničkog interfejsa sa generičkim grafičkim komponentama i kodom za validaciju argumenata i generisanje sadržaja komandne linije. Nakon kopiranja statičkog koda, generator generiše *front-end* kôd kompozicijom i konfiguracijom predefinisanih grafičkih komponenti, kao i objektni model korisničkog interfejsa, koji se koristi za validaciju argumenata i generisanje sadržaja komandne linije na osnovu unetih vrednosti. Generisanje koda se, kao i kod *CLI Generator*-a, vrši pomoću šablon-generatora.

4.7 GUI programski okvir

Programski okvir grafičkog korisničkog interfejsa je implementiran kao Electron desktop aplijacija. Aplikacija sadrži jedan prozor (prikazan na slici 3) koji prikazuje jednu ili više ulaznih formi. Svaka forma predstavlja jednu od komandi opisanog korisničkog interfejsa i sadrži ulazne elemente koji odgovaraju parametrima date komande, kao i dugmad koja vode ka formama odgovarajućih potkomandi. Aplikacija omogućuje prikaz korisničke pomoći i opisa komandi i parametara, kao i validaciju korisničkog ulaza. Pored validacije pojedinačnih parametara i komandi, validiraju se i definisani šabloni korišćenja parametara. Na posletku, aplikacija omogućava generisanje sadržaja komandne linije za opisani komandni interfejs na osnovu korisničkog unosa i slanje generiasnog sadržaja na standardni izlaz. Ovo dalje omogućava izvršavanje date komande sa odgovarajućim argumentima u interpreteru komandne linije.

Validacija ulaza i generisanje sadržaja komandne linije se vrše na generičan način, pomoću izgenerisanog JavaScript objektnog modela, koji je definisan u okviru grafičkog programskog okvira. Ovaj model opisuje strukturu korisničkog interfejsa gde su glavni koncepti, kao i kod CID modela, komande i parametri.



Slika 3: *Primer prozora grafičkog interfejsa: 1. naslov komande, 2. dugme za korisničku pomoć, 3. navigaciona traka nad-komandi, 4. opis komande, 5. parametri, 6. potkomande, 7. izgenerisani sadržaj komandne linije, 8. dugme za zatvaranje, 9. dugme za izvršavanje komande*

5. ZAKLJUČAK

Razvijeni sistem omogućava generisanje grafičkih korisničkih interfejsa i interfejsa komandne linije na osnovu jedinstvene specifikacije u jeziku specifičnom za domen. Jezik je dovoljno koncizan da omogućava brzo i jednostavno opisivanje komandi i parametara korisničkog interfejsa. Domen jezika je ograničen na generisanje komandnih interfejsa, što omogućava da pojmovi jezika budu dovoljno apstraktni da istovremeno opisuju elemente oba tipa korisničkih interfejsa. Na ovaj način se izbegava dupliciranje specifikacije i time skraćuje vreme implementacije i olakšava održavanje oba tipa korisničkih interfejsa. Sa druge strane jezik nudi fleksibilnost u opisivanju izgleda i strukture tekstualnog sadržaja komandne linije čime se omogućava opis velikog broja postojećih aplikacija sa interfejsom komandne linije i generisanje grafičkog interfejsa za iste. Takođe, jezik nudi fleksibilnost i u opisivanju strukture i rasporeda grafičkih elemenata na ulaznoj formi grafičkog korisničkog interfejsa. Pored pomenute fleksibilnosti, generatori koda imaju mogućnost određivanja smislenih podrazumevanih vrednosti za delove specifikacije koji nisu navedeni i na taj način omogućavaju korisniku da navede minimalan opis korisničkog interfejsa na osnovu koga se može izgenerisati funkcionalan kôd korisničkih interfejsa.

Takođe, pored izbegavanja dupliciranja specifikacije elemenata korisničkog interfejsa, razvijeni sistem olakšava razvoj pojedinačnih korisničkih interfejsa tako što omogućava njihov opis na višem nivou apstrakcije, na osnovu čega se generiše kôd tih interfejsa, koji inkorporira dobre prakse, dizajn šablone i standarde u oblasti dizajna korisničkih interfejsa.

Dva glavna slučaja korišćenja implementiranog sistema su generisanje grafičkog korisničkog interfejsa i interfejsa komandne linije za određenu aplikaciju i generisanje grafičkog korisničkog interfejsa za aplikaciju sa već postojećim interfejsom komandne linije. Sistem se može upotrebiti za razvoj korisničkih interfejsa za jednostavne alate koje je potrebno isporučiti klijentima koji preferiraju grafičke korisničke interfejse, a za koje takođe postoji potreba da budu automatizovani, tj. da se izvršavaju u okviru skripte za automatizaciju ili grupnu obradu.

Takođe sistem se može koristiti za razvoj grafičkih korisničkih interfejsa koji se oslanjaju na već postojeće interfejse komandne linije postojećih aplikacija, kako bi se upotreba tih aplikacija olakšala korisnicima koji iz različitih razloga izbegavaju upotrebu komandne linije. Kako je domen implementiranog sistema ograničen na generisanje jednostavnih komandnih korsničkih interfejsa, on se ne može koristiti za generisanje korisničkih interfejsa za aplikacije koje zahtevaju složenu interakciju sa korisnikom.

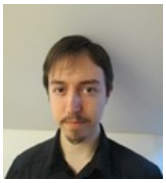
Neki od bitnih aspekata sistema kojima bi se trebala posvetiti pažnja u daljem razvoju su: testiranje, internacionalizacija generisanih korisničkih interfejsa, implementacija dodatnih ulaznih grafičkih elemenata i podrška za druge operativne sisteme (pored Microsoft Windows OS-a).

CID projekat je otvorenog koda, pod MIT licencom i dostupan na adresi: https://github.com/zeljko-bal/CID

6. LITERATURA

[1] www.igordejanovic.net/textX/
[2] jinja.pocoo.org/
[3] electronjs.org/
[4] en.wikipedia.org/wiki/Domain-specific_language
[5] en.wikipedia.org/wiki/Interface_(computing)
[6] en.wikipedia.org/wiki/User_interface
[7] en.wikipedia.org/wiki/Command-line_interface
[8] en.wikipedia.org/wiki/Graphical_user_interface
[9] www.eclipse.org/Xtext/
[10] www.eclipse.org/
[11] www.chromium.org/Home
(svim stranicama je pristupljeno u novembru 2018.)

Kratka biografija:



Željko Bal rođen je u Novom Sadu 1990. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – Softversko inženjerstvo, odbranio je 2019.god.

kontakt: zeljko.bal@gmail.com