Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.41
DOI: https://doi.org/10.24867/04BE35Zetko

NAMENSKI JEZIK I OKRUŽENJE ZA MODELOVANJE I GENERISANJE INFORMACIONIH SISTEMA OPŠTE NAMENE

A DOMAIN SPECIFIC LANGUAGE AND FRAMEWORK FOR MODELING AND GENERATION OF GENERAL PURPOSE INFORMATION SYSTEMS

Maja Zetko, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U ovom radu prezentovan je generator izvornog Java koda. Dat je pregled modela i šablona, kao i metoda i tehnika koje su korišćene za dobijanje Java koda iz modela, kao i kreiranje namenskog jezika. Takođe, navedeno je poređenje između generatora programkog koda koji su trenutno dostupni na tržištu i generatora opisanog u ovom radu.*

Ključne reči:*; Generator, definisanje šablona, model, namenski jezik*

Abstract – *In this paper, a generator of the Java source code is presented. We give a review of models and templates, as well as the methods and techniques used to obtain the Java code from a model, as well as creation of a DSL language. Also, we make a comparison of the code generators currently available in the market and the generator proposed in this paper.*

Keywords: *Generator, DSL, Template Detection, Model, Domain Specific Language*

1. UVOD

U poslednjim decenijama primećuje se povećanje potrebe za specifičnim rešenjima informacionih sistema, kao i potreba da se postojeća rešenja menjaju u skladu sa promenama u poslovanju. Klijenti imaju potrebu za novim funkcionalnostima ili modifikacijama postojećih, koje mogu da zahtevaju značajne programske izmene. U neretkim situacijama izmene su prouzrokovane zakonskim izmenama kao i razvojem savremenih tehnologija iz oblasti poslovanja za čiju primenu je neophodno izvršiti izmene u načinu poslovanja. Razvojem tehnologija stvorena je mogućnost za automatizaciju velikog dela poslovnih procesa koje zahteva korišćenje više različitih informacionih sistema, koje je potrebno integrisati.

Sve veća složenost informacionih sistema u kombinaciji sa vremenskim ograničenjem za njihov razvoj motivisala je istraživanje pristupa razvoju programskih rešenja koji su u mogućnosti da ubrzaju isporuku proizvoda i smanjuju troškove. Konkuretnost na tržištu iziskuje da se izmene u načinu poslovanja implementiraju u što je moguće kraćem roku.

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Ivan Luković, red. prof.

Model Driven Engineering (*MDE*) može doprineti bržem razvoju kvalitetnijih softverskih rešenja. U kontekstu *MDE*, model je opis sistema ili dela sistema zapisan korišćenjem dobro definisanog jezika koji podržava automatsku interpretaciju od strane računara. Model sistema na visokom nivou apstrakcije tretira se kao deo implementacije, za razliku od pristupa baziranih na modelima kod kojih se model koristio samo za potrebe analize, boljeg razumevanja i dokumentovanja sistema. Sistem predstavlja skup elemenata u interakciji i njihovih relacija. Apstrakcija je proces zanemarivanja nebitnih informacija prilikom kreiranja modela. Transformacija predstavlja automatsko generisanje ciljnog modela na osnovu početnog modela, u skladu sa definicijom transformacije. Definicija transformacije predstavlja skup pravila od kojih svako opisuje kako se jedna ili više konstrukcija iz početnog jezika može preslikati u jednu ili više konstrukcija ciljnog jezika.
Model je instanca metamodela, koji predstavlja jezik za specifikaciju modela. Metamodel se može definisati namenskim jezikom, koji je računarski jezik specijalizovan za određenu oblast primene [8]. Metamodel se specificira putem koncepata koji su definisani u meta-metamodelu. Meta-metamodel, metamodel, model sistema i instanca modela (instance klasa, korisnički podaci) formiraju četvoroslojnu arhitekturu metanivoa.
U ovom radu je analiziran pristup razvoja softvera zasnovanog na modelima prilikom izrade informacionih sistema. Proučeni su osnovni koncepti informacionih sistema opšte namene i formalno opisani namenskim jezikom. Opisan je razvoj i implementacija namenskog jezika, kao i razvoj programskog okvira za podršku modelovanju putem datog jezika. Kreiran je generator informacionog sistema zasnovan na pomenutom jeziku.
Pored navedenog, u samom master radu analizirana je strategija proširenja osnovnih funkcionalnosti generisanog informacionog sistema radi pružanja podrške u implementaciji realnih korisničkih zahteva. Proučena je opravdanost korišćenja kreiranog sistema i prateće metodologije razvoja softverskog rešenja u realnim uslovima.

2. PREGLED AKTUELNOG STANJA U OBLASTI

Na tržištu postoje generatori programskog koda čija kompleksnost u načinu korišćenja kao i dobijenog programskog koda, varira. Jedan od zastupljenijih je *Acceleo* [1], dodatak u *Eclipse* razvojnom okruženju, koji se koristi za generisanje tekstualnih programskih jezika iz

modela definisanog metamodelom Moguće je izabrati jedan od ponuđenih metamodela, dok je kreiranje specifičnog dosta jednostavno.

Kada se *Acceleo* dodatak instalira u *Eclipse*, dobije se nova *Acceleo* perspektiva i mogućnost kreiranja *Acceleo* projekta. Omogućeno je debagovanje za vreme generisanja, tako da se generisanje može pauzirati u određenom momentu kako bi se proverio dobijeni rezultat.

Jedan od jednostavnijih generatora je *Picocog* [2]. To je javno dostupna biblioteka za programski jezik *Java*. Šablon se kreira koristeći klasu *PicoWriter* i pozivajuću njenih nekoliko jednostavnih metoda za ispisivanje teksta, kao što su *writeln_r()*, *writeln_l()* i *writeln_lr()*. Moguće je instancirati više puta klasu *PicoWriter* korišćenjem metode *createDeferredWriter()* i na taj način se može pisati kôd na različitim mestima.

Pored navedenih generatora programskog koda, pomoću kojih može da se generiše kôd pisan u više programskih jezika, postoje i oni koji su specijalizovani za određeni programski jezik. Jedan od njih je *Xtend*, koji generiše *Java* kôd. Može se koristiti bilo koja od postojećih biblioteka za programski jezik *Java*. Dobijeni kôd je čitljiv i ekvivalentan je ručno pisanom *Java* programskom kodu.

Spring Boot [6] je projekat koji omogućava jednostavniji način kreiranja i konfigurisanja aplikacija koje je moguće izvršavati. Uz razvoj *Spring Boot-a* razvijen je *Spring Roo* [3], koji naredbama kroz komandnu liniju generiše potreban kôd, uključujući dodavanje potrebnih biblioteka programskog jezika *Java* kao i kreiranje potrebnih metoda. Kada se *Spring Roo* instalira, preko *CMD* komandne linije se kucaju naredbe koje generišu programski kôd. Nije potrebno kreirati model, nego se generiše jedna po jedna komponenta. Moguće je dobijeni kôd menjati u nekom od tekstualnih editora, i ta promena će biti vidljiva u komandnoj liniji. Ovim generatorom se brzo dolazi do funkcionalne aplikacije.

3. OPIS NAMENSKOG JEZIKA

Namenski jezik je računarski jezik specijalizovan za određenu oblast primene. U ovom radu je opisan namenski jezik koji sadrži sledeće koncepte:

- *Menu* - služi za generisanje strukture menija, koja može da ide u neograničenu dubinu. To se postiže tako što jedna instanca koncepta može da bude roditelj i/ili dete u odnosu na druge instance. Pri kreiranju menija prvo se uzimaju koncepti koji nemaju roditelja, i zatim njegova deca i to redosledom definisanim atributom *anOrder*. Ako je za stavku menija definisana validna oznaka objekta u atributu *anObject*, ta stavka postaje link za pristup formama i podacima vezanih za taj objekat.
- *Object* - predstavlja opis tabele čiji se podaci koriste na formi, kao i način prikazivanja podataka. Atributima koncepta se, između ostalog, definiše tabela u bazi, *where i order by* klauzule. Atributom *anPagination* se definiše da li se na listi podataka nalazi paginacija ili ne. Kada se na formi ne primenjuje paginacija, inicijalno se prikazuje prazna forma, i tek nakon izbora parametara za filtriranje i pretrage po tim parametrima, se prikazuju podaci.
- *ObjectItem* – povezan je sa konceptom *Object* i definiše kolone iz tabele koje će se prikazivati na formi aplikacije. Osnovni atribut ovog koncepta je *acType*, kojim se definiše tip podatka i način njegovog prikazivanja. *OCL* (Object Constraint Language) se koristi za definisanje obaveznih atributa u zavisnoti od tipa izabranog u *acType* atributu.
- *ObjectItemStyle* - pruža osnovne opcije koje utiču na izgled *jsp* stranica. Može se definisati *css* (Cascading Style Sheets) klase, *JavaScript* funkcije, kao i *HTML* (Hyper Text Markup Language) tagovi. Konkretne *css* klase i *JavaScript* funkcije nisu predmet ovog rada i potrebno ih je dodatno kreirati u importovati u projekat.
- *MenuLocalisation*, *ObjectLocalisation* i *ObjectItemLocalisation* – pružaju jednostavan oblik lokalizacije, pomoću koga je moguće generisati kôd aplikacije na više jezika. Navedeni koncepti su povezani sa adekvatnim osnovnim konceptom i omogućavaju da se tektualni atributu, kao što je naziv, prevedu na više jezika.
- *PrivilegeMenu*, *PrivilegeObject* i *PrivilegeObjectItem* - služe da se obezbedi da svaka grupa korisnika ima odgovrajući pristup podacima. U navedenim konceptima je za svaku grupu korisnika definisnao da li sme da vidi, menja ili dodaje podatke. Podaci o grupama korisnika i korisnicima nisu predmet ovog rada i potrebno ih je ručno implmentirati u aplikaciji.

Defnisanje konkretne sintakse podrazumeva definisanje rezervisanih reči koje će korisniku biti ponuđene i za koje će on trebati da veže konkretne vrednosti koje će se koristiti pri generisanju programskog koda. Rezervisane reči trebaju na jasan i nedvosmislen način da upute korisnika koje vrednosti treba da unese, kako bi dobijen kôd odgovarao specifikaciji funkcionalnosti aplikacije koja je cilj generisanja.

Kada se završi popunjavanje tabela, korišćenjem *SQL* jezika baze koja se koristi generiše se tekst u formatu koji zahteva namenski jezik. Izbor baze podataka koja će se koristiti zavisi od potrebe klijenta. Za potrebe ovog rada koristi se *Microsoft SQL Database* baza i *T/SQL* jezik.

Korišćenjem *Eclipse* razvojnog okruženja može se direktno kreirati tekstualni fajl u formatu koji zahteva namenski jezik. Na ovaj način razvojno okruženje nudi predefinisane reči i vodi korisnika kroz definisanje koncepata i njihovih atributa.

4. GENERISANJE PROGRAMSKOG KODA

Generatori progamskog koda su programi koji ulazne podatke prevode u programski kôd, za razliku od kompajlera, koji za rezultat daju izvršivi mašinski kôd. Važne odlike programskog koda dobijenog generisanjem su dobra čitljivost i brže testiranje, pošto se kôd dobija iz šablona koji su već formatirani i testirani.

Korišćenje *MVC2* (Model–view–controller) [4] obrazca i kreiranje višeslojnog programskog koda se pokazalao veoma pregledno i korisno. *MVC2* se obično koristi za razvoj softvera koji deli aplikaciju u tri povezana dela. Ovo se radi radi razdvajanja internih prikaza informacija sa načinom na koji su informacije predstavljene i prihvaćene od strane korisnika. Model dizajnira ove

glavne komponente koje omogućavaju efikasnu ponovnu upotrebu kodova i paralelni razvoj.

- Model – Model sadrži podatke u obliku pogodnom za konkretnu primenu[11]. Metodi za generisanje modela su prosleđeni parametri: naziv paketa, objekat za koji se generiše model, lista *ObjectItem*-a vezanih za objekat koji će biti atributi modela i lista primarnih ključeva. Modeli se generišu za sve tipove podataka koji se koriste u aplikaciji, što znači da će biti generisani za sve instance koncepta *Object*, kao i za sve instance koncepta *Choose* koji sadrže podatke o tabelama i kolonama koje se koriste u padajućim listama.
- Sloj za pristup podacima (*Data access object, DAO*) služi za direktan rad sa bazom podataka korišćenjem *Hibernate* biblioteke *Java* programskog jezika. Korišćenjem *Hibernate* anotacija nad atributima i pozivanjem *Hibernate* metoda za rad sa podacima, obezbeđuje se ispravnost podataka i koozistentno stanje baze podataka.
- Sloj za rad sa podacima (Business object *BO*) omogućava definisanje operacija koje je potrebno izvršiti nad podacima. *BO* sloj je smešten između kontrolera i *DAO* sloja, tako da kontroler komunicira samo sa *BO* slojem koji preuzima podatke iz *DAO* sloja.
- Kontroler – Kontroler ažurira model i prikaz. Unutar jedne klase može da bude više kontrolera koji su zasnovani na različitim funkcionalnostima aplikacije. Za pregled podataka se generišu dva kontrolera, jedan za *GET* poziv i jedan za *POST* poziv, dok se za otvaranje formi za izmenu podataka i za ažuriranje podatka generiše po jedan kontroler koji pozivaju odgovarajuće metode *BO* sloja.
- Komponente i komandni objekat – Pod komponentama se podarazumevaju pripereri (*Java* klase koje nasleđuju klasu *ViewPreparerSupport*) i validatori. Pripereri služe za pripremu podataka za prikaz. U njima se preuzimaju filteri sa forme, na osnovu kojih se kreira objekat za filtriranje podataka. Validatori služe za validaciju podataka preuzetih sa forme. Komandni objekti služe za jednostavnije prosleđivanje podataka na *jsp* stranicu.
- *JSP* stranice – Putem *jsp* stranice vrši se prikaz objekata u okviru aplikacije. Prva stranica koja se generiše predstavlja okvir unutar kojeg će se prikazivati lista podataka sa paginacijom. Na ovu formu se importuje stranica pod nazivom *taglib.jsp* u kojoj su odrađeni svi potrebni importi. Ova stranica je statička i ne generiše se nego se ručno kreira. Sledeće se kreira stranica na kojoj će se prikazivati lista sa podacima. Dalje se generiše stranica sa formom za pritup podacima u cilju dodavanja ili izmene.

Svaka *Java* klasa sadrži naziv paketa u kojem se nalazi. Pošto se klase različitog sloja programskog koda nalaze u različitim paketima koji ne moraju biti slični po nazivu, a cilj generatora opisanog u ovom radu je da omogući generisanje dela programskog koda koji će biti integrisan u postojeću aplikaciju, naziv paketa nije sadržan u modelu nego se prosleđuje kao ulazni parameter pri generisanju svakog sloja pojedinačno. Naziv paketa takođe predstavlja lokaciju na disku na kojoj će se sačuvati dobijeni fajl.

5. PRIMER GENERISANE APLIKACIJE

Izabran je primer aplikacije koja služi za vođenje evidencije o zadacima radnika jedne konsultantske firme. Konsultanti komuniciraju sa komitentima i zajedničkim radom rešavaju svakodnevne probleme. Kako bi produktivno iskoristili vreme potrebno je da evidentiraju zadatak tako što će opisati uočeni problem i rezervisati vreme koje je predviđeno za njegovo rešavanje.

Osnovna funkcionalnost aplikacije je vođenje evidencije o zadacima. Zadatak, u najopštijem smislu znači bilo koja aktivnost radnika u toku radnog vremena. To može biti telefonski razgovor sa klijentom, konsultacija sa kolegom, sastanak sa nadređenima, odlazak na teren i sve ostalo što zaposleni radi u toku radnog vremena. Svaki zadatak ima tip koji se bira iz padajuće liste, i može da bude jedna od prethodno opisanih stvari. Zadatak može biti otvoren, zatvoren ili na čekanju. Za svaki zadatak je potrebno popuniti opis problema koji se rešava i na kraju opisati rešenje koje je sprovedeno.

Da bi bila napravljena forma za popunjavanje predhodno opisanih podataka, prvo je potrebno kreirati tabelu u bazi podataka. Za svaki *combobox* na formi se kreira posebna tabela u bazi podataka. Za *combobox* je u strukturi namenskog jezika kreirano više koncepata, koji kada se ispravno popune generišu potrebne *SQL* izraze. Ti koncepti su:

- *Choose* – za definisanje tabele i kolone primarnog ključa
- *Value* – za definisanje kolona koje treba prikazati pri izboru, a koje mogu i ne moraju biti deo primanrog ključa
- *Where* – za definisanje *where* klauzule, koji ne mora biti popunjen
- *Order* – ze definisanje *order by* klauzule, koji ne mora biti popunjen

Kada se koristi baza podataka za kreiranje modela namenskog jezika, ti izrazi se pišu ručno.

- *SQL* izraz koji daje listu podataka - *select anUserId as 'acKey', ltrim(rtrim(acname)) + ' ' + LTRIM(rtrim(acSurname)) as 'acValue-1' from dbo.tHE_SetSubjcontact where acActive = 'T' and acUserId <> '' order by acName*
- *SQL* izraz koji daje jedan podatak na osnovu primarnog ključa - *(select ltrim(rtrim(acname)) + ' ' + ltrim(rtrim(acSurname)) from tHE_SetSubjcontact where anPrsn = anUserId) as anPrsn*

Osim navedenih, potrebno je kreirati instance koncepata *Object* za tabelu u kojoj se nalaye podaci o zadacima, *ObjectItem* sa poljima koji će se nalaziti na formi, *Ob*jectItemStyle sa odacima o izgledu jsp stranice, kao i *ObjectPrivilege,ObjectLocalisation, ObjectItemPrivilege, ObjectItemLocalisation*, koji služe za definisanje prava pristupa podacima i lokalizaciju.

Da bi se dobio meni u aplikaciji potrebno je kreirati instance koncepata *Menu, MenuPrivilege i MenuLocalisation.*

Model za predhodno opisanu aplikaciju su kreirali menadžer prodaje i *web* dizajner, uz manju pomoć

programera, što je potvrdilo da model može biti kreiran od strane krajnjeg korisnika aplikacije.
Analizom dobijenog programskog koda uočena su dva problema.
Prvi problem su predstavnjale *jsp* stranice koje nisu bile ispravne. *Jsp* stranica sa listom podataka nije bila ispravno povezana sa odgovarajućim kontrolerom, zbog čega nije bilo moguće filtriranje podataka, kao ni prelaz na stranicu za izmenu podataka. Na stranici za izmenu podataka nisu bile dodate *Java Script* funkcije definisane u modelu, zbog čega nije radila validacija i formatiranje podataka.
Drugi problem je predstavljala realizacija preslikavanja kompozitnog ključa u modelu podataka. Preslikavanje kompozitnog ključa se radi preko unutrašnje klase koja ima sufiks *PK* i koja sadrži atribute koji ulaze u kompozitni ključ. Pri generisanju unutrašnje klase, atributi nisu bili onih tipova podataka kako su definisani u modelu.
Navedeni problemi prate trenutno razvijenu verziju generatora i biće predmet rešavanja u budućem radu. Ostali delovi generisanog programskog koda su bili ispravni i nije bila potrebna dodatna korekcija. Generator programskog koda nije u potpunosti ispunio očekivanja, međutim brzina i količina dobijenog kvalitetnog koda opravdava dalja ulaganja u njegov ravoj.

6. ZAKLJUČAK

Usled povećane složenosti i brzine izmena u poslovnim procesima, kao i povećane konkurencije na tržištu, bilo je potrebno omogućiti brži razvoj informacionih sistema. Brži razvoj nije smeo da umanji kvalitet dobijenog rešenja, kao ni definisanje zahteva od strane krajnjeg korisnika.
U ovom radu je opisan razvoj i korišćenje generatora koda za programski jezik *Java*. Namenski jezik kreiran za potrebe ovog rada sadrži koncepte *Object*, *ObjectItem* i *ObjectItemStyle*, na osnovu kojih se podaci čitaju iz baze podataka i prikazuju na formi u obliku definisanom atributom *acType* koncepta *ObjectItem*. Kreirani su koncepti *Choose*, *Value*, *Where* i *Order* koji služe za generisanje dodatnih upita za preuzimanje podataka za popunjavanje combobox elemenata. Konceptima *MenuLocalisation*, *ObjectLocalisation* i *ObjectItemLocalisation* definisan je jezik dobijene aplikacije, dok su konceptima *PrivilegeMenu*, *PrivilegeObject* i *PrivilegeObjectItem* definisana prava pritupa. Generatorom se dobija višeslojan kôd koji je lako čitljiv i razumljiv.
Prednost ovog generatora u odnosu na druge dostupne na tržištu je u tome da kreiranje modela može da uradi krajnji korisnik. Na taj način se olakšava komunikacija izmeću programera i krajnjeg korisnika. Takođe je prednost što je model moguće definisati na dva načina, popunjavanjem tabela u bazi podataka i generisanjem tekstualnog fajla korišćenjem dostupnih alata izabrane baze, ili korišćenjem razvojnog okruženju *Eclipse*. Korišćenjem *Eclipse* razvojnog okruženja može se direktno kreirati tekstualni fajl u formatu koji zahteva namenski jezik. Na ovaj način razvojno okruženje nudi predefinisane reči i vodi korisnika kroz definisanje koncepata i njihovih atributa.
Programski kôd dobijen generisanjem nije potpun i potrebno ga je ručno korigovati, što nije preporučljivo. Sledeći korak u razvoju generata bi bio ispravka uočenih grešaka opisanih u prethodnim poglavljima. Zatim, je potrebno obezbediti način integracije ručno pisanog programskog koda sa generisanim, tako da se generisani kôd ne menja.
Korišćenjem namenskog jezika i generatora programskog koda opisanih u ovom radu, može se dobiti programski kôd aplikacije koji se može koristiti kao polazna tačka u razvoju složenih informacionih sistema. Iako u ovom momentu postoje nepokriveni aspekti u dobijenom programskom kodu, brzina i količina dobijenog kvalitetnog koda opravdava dalja ulaganja u njegov ravoj.
Dalji razvoj generatora programskog koda, opisanog u ovom radu, odnosio bi se na kreiranje novih tipova *ObjectItem* komponente. Kao što je navedeno, na osnovu vredosti atributa *acType* se definiše izgled i funkcionalnost polja na formi.

7. LITERATURA

[1] "Acceleo/Getting Started" https://wiki.eclipse.org/Acceleo/Getting_Started
[2] "Code Generation Is Easy - With Picocog" https://dev.to/ainslec/metaprogramming-with-picocog-44od
[3] "Spring Roo" https://docs.spring.io/spring-roo/reference/html/intro.html
[4] "Model–view–controller" https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller
[5] "MVC 2 Architectural Diagram" http://springjavatutorial.blogspot.com/2014/05/mvc1-vs-mvc2.html
[6] "UI design with Tiles and Struts", Prakash Malani 2002., https://www.javaworld.com/article/2073902/ui-design-with-tiles-and-struts.html
[7] "Uvod u MVC dizajn patern", Aleksa Vidović 2017., https://startit.rs/dizajn-paterni-mvc/
[8] "Spring Boot" https://spring.io/projects/spring-boot
[9] Model-driven Software Engineering (MDSE) in Practice. The book on MDD, MDE, MDA, MD* by Marco Brambilla, Jordi Cabot, and Manuel Wimmer. 2nd edition. Morgan & Claypool, 2017. ISBN
[10] "Domain-specific language" https://en.wikipedia.org/wiki/Domain-specific_language

Kratka biografija:

Maja Zetko rođena je u Novom Sadu 1987. Fakultet tehničkih nauka upisala je 2006. god. Bečelor rad iz oblasti Elektrotehnike i računarstva odbranila je 2011. Master studije upisala je 2016. godine.