Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.42.045
DOI: https://doi.org/10.24867/12BE22Bjekovic

# GENERATOR WEB APLIKACIJA BAZIRANIH NA ANGULARJS I SPRING RAZVOJNIM OKVIRIMA

# CODE GENERATOR FOR WEB APPLICATIONS BASED ON ANGULARJS AND SPRING FRAMEWORKS

Božo Bjeković, *Fakultet tehničkih nauka, Novi Sad*

**Oblast – ELEKTROTEHNIKA I RAČUNARSTVO**

**Kratak sadržaj –** *U radu je prikazan generator koda zadužen za generisanje osnove aplikacije kako serverske, tako i klijentske strane. Serverska strana izgenerisana je u Java programskom jeziku uz oslonac na Spring radni okvir, dok je klijentska strana u AngularJS-u. Proces generisanja bazira se na definisanom UML dijagramu modela*

**Ključne reči:** *Generator koda, UML, šabloni*

**Abstract** – *The paper presents a code generator for generating the application base, for both server and client side. The server side is generated in Java programming language based on the Spring framework, while the client side is in AngularJS. The generation process is based on a defined UML model diagram.*

**Keywords:** *Code Generator, UML, patterns*

## 1. UVOD

Primena softverskog inženjerstva nad složenim sistemima je izrazito kompleksna zbog heterogenosti sistema i izazova koji proizilaze iz različitih domena razvoja softvera. Stoga, zahtevan softver se uglavnom razvija od strane domenskih eksperata sa drugačijim tačkama gledišta, razumevanjima sistema i njegovih funkcionalnosti. Primena jezika opštih namena često ne mogu da reše ovakav problem heterogenih sistema, što dovodi do tendencije za upotrebom jezika specifičnih za domen.

Savremeni poslovni sistemi koji opisuju složene probleme postaju teži za kreiranje i za sam proces održavanja. Klijenti koji traže kompikovane zahteve sa jedne strane i zahtevni softverski projekti sa druge, nekada podrazumevaju implementaciju velikog broja povezanih domenskih entiteta.

Ručno implementiranje ovakvih sistema zahteva dosta vremena, a pored toga javljaju se i ostali problemi. Svaki programer ima svoj stil pisanja koda, što prouzrokuje nekonzistentnost u samoj implementaciji rešenja, pojava grešaka je mnogo veća u odnosu na automatizaciju procesa i migracija sistema na nove platforme je gotovo nemoguća.

Iz ovoga proizilazi potreba za kreiranjem rešenja, koje će na osnovu definicije konkretnog modela, automatizovati postupak i olakšati dalji razvoj softvera.

U ovom radu je prikazano upravo takvo rešenje, koje omogućava generisanje osnove aplikacije i njenih bazičnih funkcionalnosti u Javi i AngularJS-u na osnovu opisa modela.

Motiv za razvoj ovog rešenja javlja se kao potreba da se ubrza i olakša inicijalno kreiranje osnove jednog funkcionalnog sistema. U današnje vreme, gde broj korisnika na internetu konstantno raste, razvijanje samo serverske strane koja će pored obrade podataka prikazivati i vizuelni sadržaj korisniku, predstavlja veliko opterećenje za server. Razlozi za ovo jesu razne animacije i sitne promene koje se tiču samog korisničkog izleda, a i dalje trebaju pomoć servera. Kao odgovor na ovo, došlo je do razvoja odvojenih serverskih i klijentskih strana, gde će se klijent strana baviti samo prikazivanjem korisničkog interfejsa. Pored toga, klijentska strana je u mogućnosti da obavlja i dodatnu obradu podataka, te samim tim omogućava prebacivanje određenih funkcionalnosti sa servera, čime se vrši rasterećivanje istog. Ovim je rešen problem nepotrebnog opterećivanja servera i omogućeno razdvajanje implementacionog od prezentacionog sloja.

Međutim, sa druge strane, potrebno je više vremena za razvoj, što čini glavni motiv za razvoj generatora koda, koji će omogućiti i olakšati programerima da na osnovu kreiranog modela sistema, dobiju kreiranu osnovu na kojoj će nastaviti da razvijaju sopstvenu poslovnu logiku, bez da troše dragoceno vreme kreirajući arhitekturu sistema za obe strane, klijentsku i serversku. Serverska strana je kreirana na principu slojne arhitekture (layer architecture) sačinjene od tri sloja, gde se prvi sloj bavi logikom baze podataka, osnovnim operacijama i samom konekcijom prema bazi. Drugi sloj je zadužen za poslovnu logiku, dok je treći sloj zapravo sloj veb kontrolera. Klijentska strana je definisana na sličan način, s tim što postoje samo kontroleri, koji se bave protokom podataka i servisi koji komuniciraju sa samim serverom.

## 2. DEFINICIJA POJMOVA

### 2.1. Model

Model je apstraktni prikaz strukture, funkcije ili ponašanja sistema [1]. To je uprošćena predstava kompleksne realnosti. Svi odgovori koji se izvedu iz modela, moraju da važe i u realnom sistemu. Danas, sistemi i softver postaju veoma kompleksni i ne mogu se razumeti bez odgovarajućeg modelovanja. Njegov cilj je da uobliči na vidljiv, često formalan način ono što je suštinsko za razumevanje nekog aspekta sadržaja, strukture ili ponašanja. Nivo apstrakcije u procesu

**NAPOMENA:**
**Ovaj rad proistekao je iz master rada čiji mentor je bila dr Gordana Milosavljević, red. prof.**

modelovanja utiče na validnost modela, odnosno na uspešnost predstavljanja realnog sistema. Neki od dodatnih ciljeva modelovanja su [2]:

- Pomaže u vizuelizaciji sistema onakvog kakav jeste ili onakvog kakav želimo da bude
- Omogućava specifikaciju strukture i ponašanja sistema
- Dokumentuje odluke koje su donesene
- Obezbeđuje zajednički jezik za stejkholdere
- Omogućava jasnoću i razumevanje

### 2.2. Meta-model

Meta-model opisuje koncepte i moguću strukturu modela definišući koncepte domena i njihove veze kao i ograničenja i pravila modelovanja [1]. Proces kreiranja meta-modela naziva se metamodelovanje. Metamodelovanje se u nekim sferama ljudske aktivnosti naziva još i semantičko modelovanje, kreiranje šeme ili standardizacija domena [3]. Jezik kojim definišemo meta-modele naziva se meta-metamodel i koristi se za definisanje meta-modela, a i samog sebe, pa ih iz tih razloga se naziva samodefinišućim.

### 2.3. Platforma i transformacija

Platforma predstavlja skup podsistema i tehnologija koji pružaju koherentan skup usluga putem interfejsa i propisanih načina upotrebe, koje sve aplikacije podržane tom platformom mogu koristiti, bez potrebe da imaju saznanje o načinu implementacije pruženih funkcionalnosti [4].

Transformacija je process u kome se vrši pretvaranje jednom modela u drugi. Uvek su zasnovane na izvornom meta-modelu, jer je izvorni model tačno jedan primerak konkretnog meta-modela [1]. Pravila transformacije mogu se zasnivati samo na konstrukcijama meta-modela i ona nedvosmisleno definišu izlaz.

### 2.4. Generator koda

Generatori koda su alati koji na automatizovan način transformišu modele u kod za interpretaciju u određenoj sintaksi i time pružaju veći kvalitet softverskog rešenja i produktivnost u radu. Ovaj proces zavisi i rukovodi se meta-modelom, jezikom za modelovanje sa njegovim konceptima, semantikom i pravilima, te ulaznom sintaksom potrebnom za ciljano okruženje [5]. Generatori koda se mogu različito klasifikovati, međutim kao najčešća podela postoje deklarativne i operativne. Ova klasifikacija se zasniva na pristupu koji se koristi za određivanje generatora. U deklarativnom pristupu opisano je mapiranje između elemenata izvornog (meta-model) i ciljanog programskog jezika. Operativni pristupi, kao što su pravila transformacije grafova, definišu korake potrebne za proizvodnju ciljanog koda i datog izvornog modela.

Efikasnost generisanog koda se uglavnom ogleda u tome koliko iskusan programer ga je napisao. Glavni argument protiv generatora koda jeste tvrdnja da generisani kod ne može da ispuni stroge zahteve i efikasnost izvođenja koji su osnovni problemi pri razvoju softvera za uređaje sa ograničenom memorijom i resursima za obradu [5]. Kada kod generiše generator koji je napravio iskusni programer, uvek će proizvoditi bolji kod nego što prosečni programer piše ručno. Upravljanje memorijom, optimizacija, model programiranja i stilovi se dosledno primenjuju. U idealnom slučaju, generisani kod bi trebao izgledati kao kod koji je ručno napisao iskusni programer koji je definisao generator.

### 2.5. Jezici specifični za domen

Jezici opšte namene su mali jezici, fokusirani na određeni aspekt softverskog sistema, a ne za određenu oblast [9]. Jako teško je postići kreiranje kompletnog programa koristeći samo DSL (Domain Specific Language), međutim upotrebom više DSL-ova, uglavnom napisanih na jeziku opšte namene, možemo postići veću pokrivenost generisanog koda.

Dobro dizajnirani i kreirani jezici specifični za domen su od velike pomoći programerima, i mnogo je lakše programirati na takav način nego sa tradicionalnim bibliotekama. Ovo poboljšava produktivnost programera, što je uvek poželjno, i može poboljšati komunikaciju sa domenskim ekspertima što je takođe važan faktor u razvoju softverskog rešenja.

## 3. IMPLEMENTACIJA RJEŠENJA

Generator koda je napisan u Java programskom jeziku, za kreiranje meta-modela i konkretnog modela rešenja je zadužen MagicDraw alat i kao obrađivač šablona (template engine) je upotrebljen FreeMarker.

### 3.1. Tehnologije

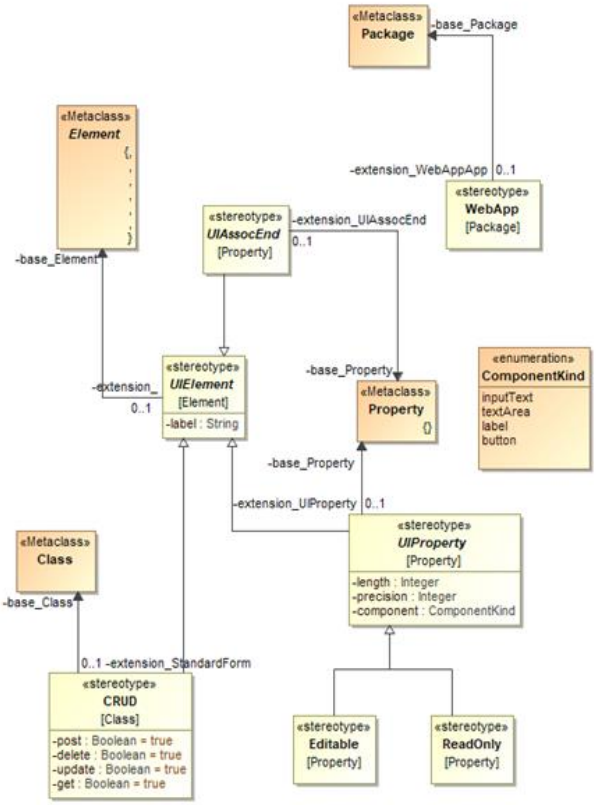
Razvojno okruženje korišćeno za razvoj i implementaciju softverskog rešenja jeste Eclipse IDE. Generator koda je razvijen i upotrebljen kao plugin unutar MagicDraw alata koji ga ujedno i snabdeva ulaznim podacima, u ovom slučaju kreiranim UML dijagramom. Kao rezultat generatora, dobija se odvojena server (back-end) i klijent (front-end) aplikacija. Server aplikacija je izgenerisana u programskom jeziku Java, oslanjajući se na Spring framework, dok je klijentska aplikacija kreirana uz pomoć AngularJS jezika.

### 3.2. Meta-model

Meta-model, prikazan na slici 1, predstavlja osnovu na kojoj je kreiran model, čija struktura će biti objašnjena detaljnije u narednom poglavlju. Osnovni model proširen je stereotipima (stereotype) koji definišu kako se postojeća meta-klasa može nadograditi. Stereotipi se mogu kombinovati i mogu se primeniti na bilo koji elemenat modela (aktivnostima, atributima, zavisnostima i drugim). WebApp stereotip koji nasleđuje Package element predstavlja glavni paket koji grupiše sve klase i druge elemente modela zajedno. Svaki dijagram mora da se nađe unutar jednog paketa. UIElement stereotip, koji nasleđuje Element klasu meta-modela, proširuje element sa novim poljem label i predstavlja komponentu korisničkog interfejsa sa svojim nazivom. CRUD (Create Read Update Delete) stereotip nasleđuje Class elemenat meta-modela i predstavlja opis mogućih operacija nad primenjenim entitetom. Polja koja se nalaze unutar njega su: post (mogućnost kreiranja entiteta), delete (mogućnost

brisanja entiteta), update (mogućnost izmene entiteta) i get (mogućnost čitanja entiteta). Sva polja unutar pomenutog stereotipa su opciona i prilikom generisanja koda, na osnovu ovih polja određuje se koji tipovi metoda će biti kreirani. UIProperty stereotip nasleđuje Property element meta-modela i definiše opis polja nad kojim se primenjuje. Definiše dužinu (length polje), preciznost (precision polje) i tip polja (component polje). Tip polja je označen enumeracijom ComponentKind i definiše moguće tipove polja. U ovo slučaju to su tekstualno polje (inputText i textArea), oznaka (label) i dugme (button). UIProperty stereotip je dodatno proširen sa dva stereotipa Editable i ReadOnly. Editable stereotip označava da li neko polje ima mogućnost izmene, dok sa druge strane ReadOnly označava zabranu izmene.

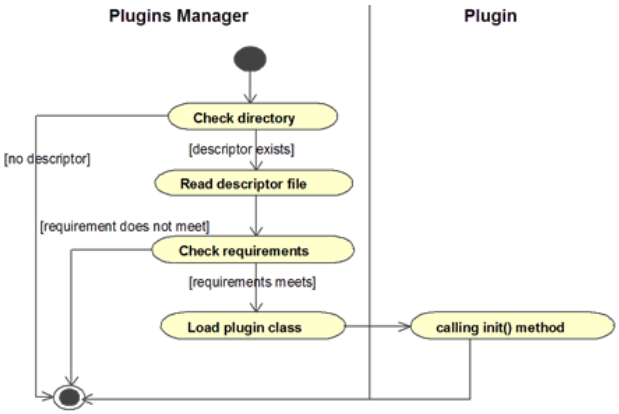
Opisani meta-model se dalje koristi prilikom kreiranja konkretnog modela veb aplikacije i na osnovu njega se proverava ispravnost. MagicDraw alat vrši proveru ispravnosti kreiranog modela na osnovu zadatog meta-modela. Prikazani meta-model se može koristiti za više tipova jednostavnih veb aplikacija koje podržavaju CRUD operacije nad modelima. Jedan od takvih primera aplikacije jeste veb forum, gde će se na osnovu kreiranog modela generisati projekat sa celokupnom podrškom za CRUD operacije, sto će biti prikazano i objašnjeno u narednom poglavlju.




Slika 1. Meta-model jezika za opis definicije podataka

### 3.3. Implementacija generatora

Kreiranje i dodavanje plugin-a su način na koji se može proširiti i promeniti funkcionalnost samog MagicDraw alata, što jeste i glavna svrha plugin arhitekture. Na slici 2 prikazan je način provere i dodavanja novog plugina u MagicDraw alat. Prilikom pokretanja alata, pretražuje se plugin direktorijum i pokušava se pronaći plugin descriptor kojeg čita plugin menadžer. Ukoliko je sve ispravno navedeno plugn manadžer poziva init() metodu koja je obavezna u kreiranom plugin-u.




Slika 2. Način dodavanja plugin-a

Unutar pomenute metode, vrši se dodavanje novog podmenija unutar glavnog menija MagicDraw-a i registrovanje svih neophodnih generatora modula koji će biti korišćeni. Generatori modula se registruju uz odgovarajuće parametre koji se prosleđuju u vidu ključeva na osnovu čega se dobijaju konkretne vrednosti iz XML zapisa. Svaki generator modula je zadužen za generisanje odgovarajućeg dela aplikacije i postoje sledeći:

- Generator modela
- Generator enumeracija
- Generator sloja za pristup podacima (Repository layer)
- Generator poslovne logike (Service layer)
- Generator kontrolera (Controller layer)
- Generator AngularJS kontrolera
- Generator AngularJS servisa
- Generator HTML stranica
- Generator AngularJS rutiranja
- Generator početne stranice

Nakon kreiranog modela, u ovom slučaju veb foruma, i nakon njegove provere ispravnosti unutar MagicDraw alata, sledi proces generisanja izvršnog koda. Proces se sastoji od rednog pozivanja generatora modula koji su zaduženi sa odgovarajuće delove aplikacije. Klikom na generate dugme, unutar dodatog pomenija strartuje se process generisanja. Generator koda, kreiran kao plugin, preuzima model iz MagicDraw alata i parsira ga. U toku parsiranja pomenutog modela, izdvajaju se i prepoznaju odgovarajući elementi, poput klasa, enumeracija, paketa, atributa i drugih.

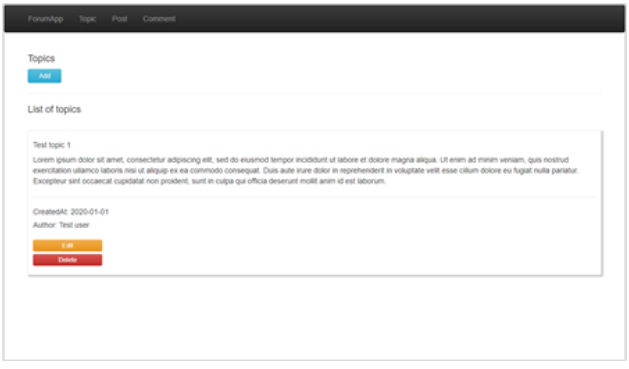
Prolaskom kroz sve elemente modela proverava se tip i na osnovu njega kreira se odgovarajuća instanca klase. Postoje tri provere tipa elemenata za proveru i to su klasa, enumeracija i paket. Ukoliko se radi o klasi, prvo se proverava da li pripada odgovarajućem paketu. Ako je ovaj uslov ispunjen, sledi proces kreiranja klase sa svim potrebnim zavisnostima, poljima i metodama.

Na listingu 1 prikazan je deo šablona koji se koristi u procesu generisanja entiteta i odnosi se na generisanje polja. U odnosu na tip veze, generiše se odgovarajući tip polja u izvornom kodu.

```
<#list properties as property>
    <#if property.upper == 1 >
    public ${property.type} get${property.name?cap_first}
    (){
        return ${property.name};
    }

    public void
        set${property.name?cap_first}(${property.type}
        ${property.name}){
           this.${property.name} = ${property.name};
    }

    <#elseif property.upper == -1 >
    public Set<${property.type}>
        get${property.name?cap_first}(){
        return ${property.name};
    }

    public void set${property.name?cap_first}(
        Set<${property.type}> ${property.name}){
        this.${property.name} = ${property.name};
    }

    <#else>
        <#list 1..property.upper as i>
    public ${property.type}
        get${property.name?cap_first}${i}(){
        return ${property.name}${i};
    }

    public void
        set${property.name?cap_first}${i}(${property.type}
        ${property.name}${i}){
        this.${property.name}${i} = ${property.name}${i};
    }

        </#list>
    </#if>
</#list>
```

Listing 1. Primer FreeMarker šablona

Prilikom pokretanja aplikacije uočava se traka za navigaciju koja prikazuje navedeno ime aplikacije, u ovom slučaju veb foruma. Pored toga, navedeni su svi entiteti, kao linkovi, kreirani unutar UML dijagrama modela pomoću kojih se vrši navigacija unutar aplikacije. Na prvom ekranu prikaza je strana teme (Topic), sa naslovom i dugmetom za kreiranje nove. Pored toga, ostatak ekrana namenjen je za prikaz kreiranih tema, njihovu izmenu i brisanje. Klikom na dugme "Add" otvara se modalni dijalog sa formom za kreiranje nove teme.




Slika 3. Prikaz dodog entiteta

## 4. ZAKLJUČAK

Zadatak ovog rada bio je da prikaže način kreiranja osnove funkcionalnog sistema i njegove arhitekture na osnovu definisanog modela. S obzirom da se u današnje vreme razvoj veb aplikacija svodi na kreiranje dva odvojena dela, serverski koji se bavi obradom podataka i klijentski koji se bavi prikazom, programeri trebaju više vremena kako bi kreirali osnove i postavili projekat za obe strane. Rešenjem, tj. generatorom koda prikazanim u ovom radu, omogućava se programerima da na osnovu kreiranog modela sistema dobiju postavljenu osnovu za obe strane i fokusiraju se na kreiranje konkretne logike aplikacije.

Pod ovim se podrazumeva da serverska strana poseduje kreirane entitete, slojnu arhitekturu koja je podeljena na deo upravljanja bazom podataka, deo poslovne logike sistema i deo kontrolera, odnosno sam veb API.

Pored ovoga, napravljene su i sve navedene CRUD operacije naznačene u procesu modelovanja, upotrebom MagicDraw alata. Klijentska strana, na osnovu kreiranog modela, sadrži generisane HTML stranice za svaki entitet, kontrolere zadužene za kontrolisanje podataka i servise spremne za komunikaciju sa serverskom stranom. HTML stranice poseduju forme koje služe za kreiranje i izmenu entiteta, kao i prikaz svih postojećih entiteta. Servisi imaju odgovarajuće metode, takođe generisane na osnovu naznačenih CRUD operacija unutar modela.

Postoji mnogo prostora za dalje razvijanje i unapređenje ovog generatora. Neki od pravaca mogu biti proširenje rešenja da podržava generisanje u drugim programskim jezicima i platformama, što se odnosi na obe strane. Poboljšanje meta-modela u cilju podrške dodatnih koncepata, jeste isto jedan od pravaca. Takođe, postoji mogućnost razvoja dodatnog generatora koji bi se, pored komunikacije sa osnovnim, striktno bavio generisanjem korisničkog interfejsa i na taj način bi se dobila celina u kojoj bi se programeri još više usmereni na konkretno pisanje poslovne logike.

## 5. LITERATURA


[1] M. Voelter, T. Stahl, Model-Driven Software Development: Technology, Engineering, Management, John Wiley & Sons, 2006, Foreword.

[2] D. Z. Jeremic, "Uvod u modelovanje korišćenjem UML-a, https://www.vps.ns.ac.rs/Materijal/mat22111.pdf.

[3] I. Dejanović, Prilog metodama brzog razvoja softvera na bazi proširivih jezičkih specifikacija, 2011.

[4] G. Milosavljević, Prezentacije sa predavanja predmeta Metodologije brzog razvoja softvera.

[5] S. Kelly and J.-P. Tolvanen, Domain-Specific Modeling, Wiley-IEEE Computer Society Pr, 2008.


**Kratka biografija:**

**Božo Bjeković** rođen je 31.05.1994. godine u Trebinju. Osnovnu školu „Sveti Sava", u Gacku, završio je 2009. godine. Gimnaziju „Pero Slijepčević" u Gacku završio je 2013. godine. Iste godine upisao je Fakultet tehničkih nauka u Novom Sadu, smer Softversko inženjerstvo i informacione tehnologije. Osnovne akademske studije završio je u oktobru 2017. godine. Iste godine je upisao master akademske studije na istoimenom smeru.