Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.41
DOI: https://doi.org/10.24867/05BE13Simic

GENERISANJE KODA ZA VIZUALIZACIJU REZULTATA ELEKTROENERGETSKIH PRORAČUNA UZ UPOTREBU OBLASNO SPECIFIČNOG JEZIKA

CODE GENERATION FOR VISUALIZING THE RESULTS OF THE ELECTRIC POWER CALCULATIONS USING A DOMAIN SPECIFIC LANGUAGE

Vladislav Simić, *Fakultet tehničkih nauka, Novi Sad*

Oblast – ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *U radu je predstavljena analiza mogućnosti stvaranja jezika kojim se modeluje izvještaj za rezultate proračuna elektroenergetskih funkcija kao i generisanje izvornog koda na osnovu modela tog izvještaja. Analizom softvera za menadžment elektroenergetskih sistema usvojeni su koncepti na kojima se zasniva domenski specifičan jezik za modelovanje izvještaja rezultata proračuna. Rješenje opisano u radu implementirano je uz oslonac na Microsoft DSL tehnologiju. Cilj ponuđenog rješenja je da olakša i ubrza razvoj segmenata kompleksnog elektroenergetskog softvera, odnosno razvoj izvještaja za rezultate proračuna.*

Ključne reči: *Modelovanje, metamodel, domenski specifičan jezik, generator koda, UML, DSL*

Abstract – *The paper presents an analysis of the possibility of creating a language which models the report for the results of the calculation of electrical energy functions as well as generating the source code based on the model of that report. An analysis of the software for the management of electric power systems has provided concepts on which the domain specific language for the modeling of the report on results of the calculations is based. The solution described in the paper is implemented with the support of Microsoft DSL technology. The goal of the offered solution is to facilitate and accelerate the development of segments of complex power software, that is, the development of the report on the results of the calculations.*

Keywords: *Modeling, metamodel, domain specific language, code generator, UML, DSL*

1. UVOD

Jedan od osnovnih zadataka prilikom implementacije softvera predstavlja odabir programskog jezika. Na njegov izbor utiču mnogobrojni faktori kao što su poznavanje samih tehnologija, kompanijska pravila i procedure, kao i implementacioni zahtjevi koji se očekuju od rješenja. Programski jezici predstavljaju osnovni alat za razvoj softvera i dijele se na jezike opšte namjene i jezike specifične za domen [1].

Druga vrsta izazova koja se javlja prilikom razvoja softvera predstavlja proces modelovanja rješenja domenskog problema. U većini slučajeva, koncepti jezika opšte namjene nisu usklađeni sa konceptima domena problema. To ima za posljedicu složenije modelovanje rješenja problema kao i samu implementaciju rješenja. Sa druge strane, korištenjem jezika specifičnog za domen omogućeno je trivijalno modelovanje rješenja, obzirom da je jezik zasnovan na konceptima domena za koji se projektuje [2].

Tema ovog rada jeste analiza vezana za mogućnost implementiranja domenski specifičnog jezika kojim je moguće modelovati u tekstualnoj ili grafičkoj formi izvještaje za rezultate proračuna elektroenergetskih funkcija [3]. Sa druge strane rad se bavi i analizom mogućnosti implementiranja generatora koda koji koristi koncepte prethodno pomenutog jezika.

2. POSTAVKA PROBLEMA

Razlog za bavljenje ovim problemom proizilazi iz potrebe da se implementira univerzalno rješenje pomoću koga bi domenski eksperti na relativno trivijalan način projektovali izvještaje rezultata elektroenergetskih proračuna. Rezultati koji se prikazuju krajnjem korisniku organizovani su u sličnim formama posmatrajući različite funkcije. To znači da prikaz rezultata svake funkcije projektujemo koristeći iste elemente grafičkog korisničkog interfejsa.

Analogno načinu prikazivanja rezultata i izvorni kod ima identičnu strukturu za svaki od tipova elementa elektroenergetskog sistema. Izvorni kod kojim je implementiran softver je uglavnom repetitivan i generičan pri posmatranju određenih segmenata.

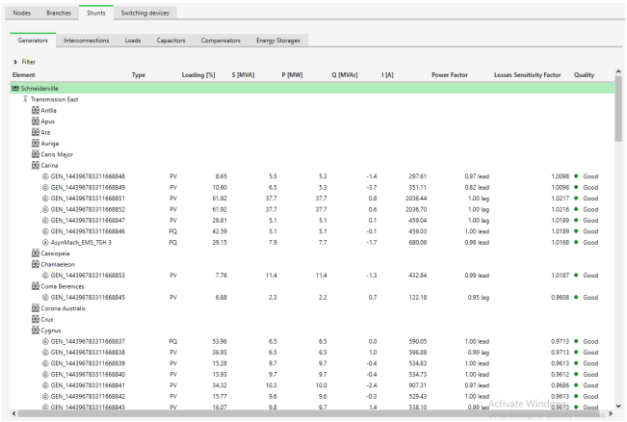
Na osnovu toga, uvođenjem domenski specifičnog jezika olakšalo bi se modelovanje izvještaja, a uvođenjem generatora koda koji se oslanja na koncepte tog jezika omogućila bi se automatska transformacija modela na izvorni kod i izbjeglo bi se manuelno kodiranje segmenata koda koji su repetitivni i generični. Bitno je napomenuti da kada se kaže da je izvorni kod repetitivan ne misli se da na više mjesta postoji identičan kod, već da su različiti elektroenergetski elementi opisani različitim klasama koje imaj istu strukturu koda, dok njihova svojstva definišu elementi za koje su klase kreirane.

Neke od osnovnih funkcija za elektroenergetske proračune su: proračuni tokova snaga, estimacija stanja, proračuni optimalnih tokova snaga, analiza optimalne topološke promjene, analiza sigurnosti prilikom ispada i druge [3]. Sa druge strane, osnovne elemenate elektroenergskog

---

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Nemanja Nedić, docent.

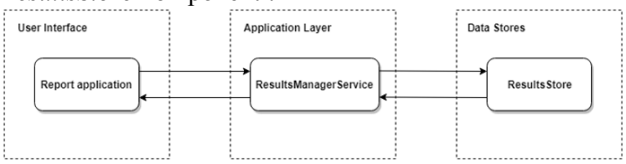
sistema predstavljaju generatori, vodovi, transformatori, kondenzatori, potrošači [4]. Primjera radi, posmatrajući generator, parametri koji se proračunavaju nad njim su aktivna, reaktivna i prividna snaga [4]. Rezultati proračuna ovih parametara u najjednostavnijem obliku korisniku se prikazuju tako što se rezultati za svaki tip elementa organizuju u kartice (eng. *Tab*), a u okviru svake kartice nalazi se tabela, gdje kolone tabele predstavljaju parametre elemenata. Takav jedan primjer prikazivanja rezultata predstavljen je na slici 2.1.



Slika 2.1 *Prikaz rezultata proračuna*

3. ANALIZA DOMENA

Softver koji omogućava izvršavanje elektroenergetskih proračuna i za koji se projektuje ovaj generator koda je implementiran u troslojnoj klijent-server arhitekturi (slika 3.1). U prezentacionom sloju nalazi se *Report* aplikacija koja prikazuje rezultate proračuna. U srednjem ili aplikativnom sloju nalazi se *ResultsManagerService*. Ova komponenta na zahtjev prezentacionog sloja dobavlja rezultate iz skladišta i u određenoj strukturi ih šalje *Report* aplikaciji. Skladištenje podataka povjereno je *ResultsStore* komponenti.



Slika 3.1 *Arhitektura softvera*

Rezultati proračuna za svaki elektroenergetski element modelovani su zasebnom klasom. Vrijednosti atributa u okviru ovih klasa predstavljaju rezultate proračuna i čuvaju se u okviru *ResultsStore* komponente. Pristup tim vrijednostima omogućen je upotrebom *Generic Data Access* (*GDA)* interfejsa koji *ResultsStore* komponenta izlaže drugim komponentama. *GDA* interfejs omogućava pristup podacima bez ikakvog poznavanja logičke šeme baze podataka [5]. Da bi *GDA* „znao“ da pristupi određenoj klasi ili njenom atributom, potrebno je na određeni način svaku klasu ili njen atribut jednoznačno odrediti. To se postiže uvođenjem jedinstvenog identifikatora, tzv. *ModelIdentificator*-a koji predstavlja 64-bitnu vrijednost. Mapiranjem jedinstvene 64-bitne vrijednost na određenu klasu ili atribut klase dobijaju se metapodaci koji opisuju konkretne podatke. Fomiranjem *ModelIdentificator*-a prema predefinisanim pravilima moguće je opisati hijerahiju nasleđivanja između klasa kao i organizaciju atributa unutar klase. Kako su podaci modelovani korištenjem identifikatora, a u *GDA* upitu treba da se nađe generički identifikator, tako se i pristupanje određenoj klasi ili vrijednosti atributa klase obavlja navođenjem identifikatora.

Uloga *ResultsManagerService*-a je da dobavi rezultate korištenjem *GDA* upita i da ih u odgovarajućoj izlaznoj strukturi proslijedi prezentacionom sloju. Definisanje te strukture postiže se definisanjem klasa modela koje modeluju različite elektroenergetske elemente. Za primjer se mogu uzeti klase koje modeluju kondenzatore. Klasa *EMSLoadFlowCapacitorRecord* (slika 3.2) sa pripadajućim atributima modeluje svojstva kondenzatora. Klasa *EMSLoadFlowCapacitorResult* (slika 3.3) predstavlja enkapsulaciju klase *EMSLoadFlowCapacitorRecord*. Objekat ove klase sadrži listu objekata klase *EMSLoad-FlowCapacitorRecord*. Klasa *EMSLoadFlowResult* (slika 3.4) predstavlja nadklasu za sve klase koje opisuju rezultate. Instanca ove klase prenosi se na prezentacioni sloj putem *WCF* radnog okvira.

```
[DataContract]
0 references
public class EMSLoadFlowCapacitorRecord
{
    #region Properties

    [DataMember]
    0 references
    public float VoltageLevel { get; set; }

    [DataMember]
    0 references
    public float Voltage { get; set; }
}
```

Slika 3.2 Klasa *EMSLoadFlowCapacitorRecord*

```
[DataContract]
1 reference
public class EMSLoadFlowCapacitorResult : EMSLoadFlowReportResult
{
    #region Constructors

    0 references
    public EMSLoadFlowCapacitorResult()
    {
        EMSLoadFlowReportType = EMSLoadFlowReportType.Capacitor;
    }

    #endregion Constructors

    #region Properties

    [DataMember]
    0 references
    public List<EMSLoadFlowCapacitorRecord> Records { get; set; }

    #endregion Properties
}
```

Slika 3.3 Klasa *EMSLoadFlowCapacitorResult*

```
[DataContract]
[KnownType(typeof(EMSLoadFlowGeneratorResult))]
[KnownType(typeof(EMSLoadFlowCapacitorResult))]
2 references
public class EMSLoadFlowResult : JobResult
{
    #region Constructors

    0 references
    public EMSLoadFlowResult()
    {
    }

    0 references
    public EMSLoadFlowResult(EMSLoadFlowReportType reportType)
    {
        EMSLoadFlowReportType = reportType;
    }

    #endregion Constructors

    #region Properties

    [DataMember]
    1 reference
    public EMSLoadFlowReportType EMSLoadFlowReportType { get; set; }

    #endregion Properties
}
```

Slika 3.4 Klasa *EMSLoadFlowResult*

Uloga prezentacionog sloja (*UI* komponenta) je direktna interakcija sa korisnikom, zatim slanje zahtjeva za dobavljanje rezultata ka *ResultsManagerService*-u i obrada rezultata koje servis vrati. Kod na *UI* komponenti je napisan u skladu sa *MVVM* [6] šablonom koji podrazumijeva postojanje tri komponente, a to su model podataka, prezentacioni dio (*view*) i tzv. *view-model* dio koji predstavlja spregu između modela i prezentacionog dijela.

Implementacija prezentacionog dijela napisana je upotrebom *WPF* tehnologije [7]. Prema *WPF*-u, svaka view klasa sastoji se od dijela koda koji je pisan u *XAML* jeziku i dijela koda koji je pisan u C# jeziku koji definiše ponašanje elemenata koji su prethodno definisani u *XAML*-u (tzv. *code-behind*). *XAML* je baziran na *XML* deklarativnom jeziku, koji se koristi za opis izgleda aplikacije. Sav kod napisan u *XAML*-u, predstavljen je kao hijerarhija ugnježdenih elemenata, poznatijem kao stablo elemenata. U slučaju izvještaja za rezultate proračune, koji je predmet ovog rada, elementi koji figurišu u *XAML*-u su *Grid*, *TabControl*, *TabItem*, *DataGrid* i *DataGridColumn*.

View-model klase predstavljaju sponu između modela i *view* komponenti. U okviru ovih klasa nalaze se podaci neophodni za funkcionisanje prezentacionog dijela. Primjer *view-model* klase za kondenzatore prikazan je na slici 3.5.

```
public class EMSLoadFlowCapacitorViewModel
{
    #region Fields

    private EMSLoadFlowCapacitorRecord reportRecord;

    #endregion Fields

    #region Constructors

    --references
    public EMSLoadFlowCapacitorViewModel()
    {
    }

    --references
    public EMSLoadFlowCapacitorViewModel(EMSLoadFlowCapacitorRecord reportRecord)
    {
        if (reportRecord == null)
        {
            return;
        }
        this.reportRecord = reportRecord;
    }

    #endregion Constructors

    #region Properties

    --references
    public float? Voltage
    {
        get
        {
            return UnitConverterHelper.ConvertFromDMS(MeasurementType.Voltage, reportRecord.Voltage);
        }
    }

    --references
    public float? Voltagelevel
    {
        get
        {
            return UnitConverterHelper.ConvertFromDMS(MeasurementType.Voltage, reportRecord.VoltageLevel);
        }
    }

    #endregion Properties
}
```

Slika 3.5 *View-model* klasa

4. OPIS RJEŠENJA

Specifikacija metamodela implementirana je uz oslonac na *Microsoft DSL* tehnologiju [8]. Generator koda implementiran je upotrebom T4 tekstualnih šablona [9] koji referencija dati metamodel, koriste njegove koncepte i specificiraju procese generisanja koda na osnovu tih koncepata.

3.1. ReportDSL metamodel

Specifikacija metamodela definiše tri vrste veza između elemenata modela [8] [10]:

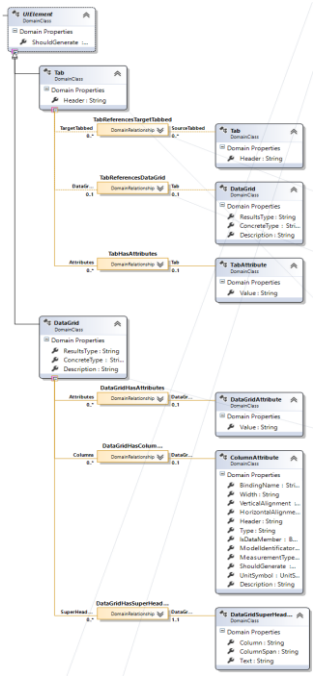
- *EmbeddingRelationship* – veza ugrađivanja koja predstavlja takav vid relacije da su elementi na nižem nivou ugrađeni u element na višem nivou,
- *ReferenceRelationship* – veza tipa referenca predstavlja vezu gdje su elementi na nižem nivou samo referencirani od strane elementa na višem nivou,
- *Inheritance* – veza nasleđivanja, gdje elementi preuzimaju osobine od elementa kojeg nasleđuju.

Svi elementi koji imaju ime, direktno ili indirektno nasleđuju klasu "*NamedElement*" koja ima atribut *Name*. Metamodel specificira apstraktnu metaklasu *UIElement* (slika 3.6) koju nasleđuju metaklase *Tab* i *DataGrid*. Ove dvije metaklase sa svojim atributima modeluju *XAML* kontrole.

Metaklasa *Tab* (slika 3.6) vezom reference ka metaklasi *Tab* specificira da *Tab* može da bude u relaciji sa nula ili više drugih *Tab*-ova. Relaciju nula ili jedan ima ka *DataGrid*-u, što znači da *Tab* može, a i ne mora da ima relaciju ka *DataGrid*-u. *Tab* ima definisanu vezu ugrađivanja "*TabHasAttributes*" ka metaklasi *TabAttribute*. Ovom vezom specificiramo da se unutar instance tipa *Tab* može naći više atributa koji je opisuju. Na osnovu vrijednosti tih atributa generišu se odgovarajući *XAML* atributi koji opisuju kontrolu *Tab*.

Metaklasa *DataGrid* (slika 3.6) posjeduje posjeduje tri veze ugrađivanja. Veza ugrađivanja "*DataGridHasAttributes*" ka metaklasi *DataGridAttribute* specificira se da u okviru *DataGrid*-a može naći više atributa koji ga opisuju. Na osnovu vrijednosti ovih atributa generišu se odgovarajući *XAML* atributi. Veza ugrađivanja "*DataGridHasColumns*" specificira da jedna instanca tipa *DataGrid* sadrži nula ili više *DataGrid* kolona. Veza ugrađivanja *DataGridHasSuperHeaders* specificira da se u *DataGrid*-u može naći više instanci tipa *DataGridSuperHeader* koje modeluju nadnaslove za odgovarajuće kolone. *DataGrid* posjeduje sledeće atribute:

- *ResultsType* – naziv klase koja opisuje svojstva elektroenergetskog elementa,
- *ConcreteType* – naziv klase koja opisuje rezultate za određeni elektroenergetski element (iz enumeracije koja sadrži spisak klasa modela koje se mogu instancirati),
- *Description* – opis.



Slika 3.6 Metaklase *Tab* i *DataGrid*

Metaklasa *ColumnAttribute* modeluje *DataGrid* kolonu. Posjeduje sledeće atribute:

- *BindingName* – vrijednost atributa se generiše u *XAML* kodu za istoimeni atributi,
- *IsDataMember* – označava da li će se atribut serijalizovati,
- *ModelIdentificator* – tekstualna vrijednost sa nazivom istoimenog atributa,
- *MeasurementType* – naziv mjerene fizičke veličine za koju data kolona prikazuje rezultate proračuna
- *UnitSymbol* – naziv fizičke veličine za koju data kolona prikazuje rezultate proračuna,
- *ShouldGenerate* – označava da li će se data kolona, tj atribut u modelima generisati,
- *Description* – opis kolone (klase),
- *XAML* atributi (*Width*, *VerticalAlignment*, *HorizontalAlignment*, *Header*) – čija se vrijednost direktno mapira na vrijednosti u *XAML* kodu,
- *Type* – ovim svojstvom se specificira tip podataka koji će imati vrijednosti kolone.

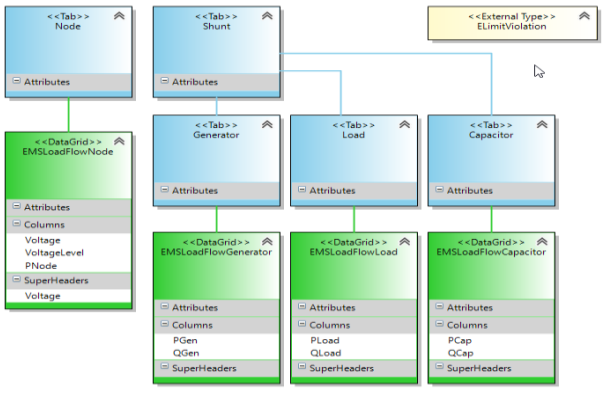
Metaklasa *SuperHeader* modeluje nadnaslov (zaglavlje) za odgovarajuće kolone. Posjeduje sledeće atribute:

- *Column* – specificira koja je prva kolona koja će biti pod datim nadnaslovom,
- *ColumnSpan* – specificira koliko kolona će se naći u okviru datog nadnaslova,
- *Text* – definiše tekst koji će imati nadnaslov.

3.1. Generisanje koda

Svaki od T4 tekstualnih šablona [9] posjeduje odgovarajuću programsku logiku pomoću koje se generiše novi tekstualni fajl koji posjeduje određenu sintaksu, a u slučaju ovog rada to je izvorni kod. U toku procesa generisanja koda, prolazi se kroz svaki od šablona i na osnovnu programske logike definiše se sadržaj generisanog izvornog koda.

Programska logika je potpomognuta ulaznim podacima koji se dobijaju sa dijagrama koji je modelovan pomoću *ReportDSL* jezika. Svaki od šablona za ulazni podatak dobija instancu modela dijagrama i u mogućnosti je da iterira kroz sve tipove elemenata sa dijagrama, pa na osnovu implementacije šablona obrađuje se element sa dijagrama za koji je šablon napisan. Primjer jednog modela izvještaja za rezultate proračuna funkcije za optimalne tokove snaga prikazan je na slici 3.7.



Slika 3.7 *Primjer modela izvještaja*

5. ZAKLJUČAK

Pretpostavka od koje se krenulo u implementaciju ovog rada pokazala se tačnom. Za izvještaj i izvorni kod koji je analiziran u ovom radu moguće je implementirati metamodel, odnosno generator koda, takav da se na relativno trivijalan način modeluje dati izvještaj i na osnovu tog modela generiše izvorni kod. Rezultat ove implementacije donio je mnogobrojne prednosti u razvoju sofvera, a najvažnije od njih su smanjanje procenta unosa ljudske greške obzirom da se kod automatski generiše na osnovu modela, čist i pregledan kod i na kraju višestruka vremenska ušteda.

6. LITERATURA

[1] I. Dejanović, Prilog metodama brzog razvoja softvera na bazi proširivih jezičkih specifikacija, Novi Sad: Fakultet tehničkih nauka, 2011.

[2] I. Dejanović, Metamodel, editor modela i generator poslovnih aplikacija, Novi Sad: Fakultet tehničkih nauka, 2008.

[3] V. Strezoski, D. Popović, N. Katić, G. Švenda, Z. Gorečan, J. Dujić / D. Bekut, Osnovne energetske funkcije za analizu, upravljanje i planiranje pogona srednjenaponskih distributivnih mreža, Kopaonik: IV skup Trendovi Razvoja: Nove tehnologije u elektrodistribuciji, 1998.

[4] V.Strezoski, Osnovi elektroenergetike, Novi Sad: Fakultet tehničkih nauka, 2014.

[5] IEC: 61970 403 Ed.1: Energy management system application program interface (EMS API), 2008.

[6] M. Docs, „The MVVM Pattern,“ dostupno na: https://docs.microsoft.com/. [datum pristupa 03 06 2019].

[7] M. Docs, „Windows Presentation Foundation,“ dostupno na: https://docs.microsoft.com/. [datum pristupa 17 06 2019].

[8] M. Docs, „Understanding Models, Classes and Relationships,“ dostupno na: https://docs.microsoft.com. [datum pristupa 03 06 2019].

[9] M. Docs, „Code Generation and T4 Text Templates,“ dostupno na: https://docs.microsoft.com/. [datum pristupa 03 06 2019].

[10] J.-P. T. S. Kelly, Domain Specific Modeling: Enabling Full Code Generation,, Hoboken, New Jersey: Wiley IEEE Computer Society Pr, 2008..

Kratka biografija

Vladislav Simić rođen je u Bijeljini 02.02.1994. godine. Diplomirao je na osnovnim studijama elektrotehnike i računarstva na Fakultetu tehničkih nauka u Novom Sadu 2017. godine, a master rad je obranio 2019. godine na istom fakultetu. Kontakt: dovlasim@gmail.com