Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/05TI01Lalosevic

JEZIK I GENERATOR KODA ZA AUTOMATIZACIJU RAZVOJA UZAJAMNO ZAVISNIH DOKUMENATA U OKVIRU SOFTVERSKIH PROIZVODNIH LINIJA

LANGUAGE AND CODE GENERATOR FOR AUTOMATING THE DEVELOPMENT OF MUTUALLY DEPENDENT DOCUMENTS IN SOFTWARE PRODUCT LINES

Tijana Lalošević, *Fakultet tehničkih nauka, Novi Sad*

Oblast – INFORMACIONI INŽENJERING

Kratak sadržaj – *U ovom radu predstavljen je jezik i generator koda za automatizaciju razvoja uzajamno zavisnih dokumenata u okviru softverskih proizvodnih linija (SPL). Model se zadaje kroz grafički korisnički interfejs generatorske aplikacije. Generisanje koda se vrši uz pomoć obrađivača šablona, kao i direktnom manipulacijom sintaksnih stabala korišćenjem parsera za jezike koji se koriste na ciljnoj platformi.*

Ključne reči: *Generator koda, DSL, SPL, šablon, metapodaci*

Abstract – *This paper presents the language and code generator for the automation of the development of mutually dependent documents within software product lines (SPL). The model is created using generator application's user interface. Code generation is performed using template engines and also by direct manipulation of syntax trees, using parsers for target platform languages.*

Ključne reči: *Code generator, DSL, SPL, template, metadata*

1. UVOD

Softverska industrija je doživela ekspanziju poslednjih dvadesetak godina. Dr. William Raduchel, profesor na Harvardu i izvršni direktor u kompaniji Sun Microsystems, Xerox i AOL Time Warner, opisao je softver kao „srž većine modernih organizacija, proizvoda i usluga“ [1]. Tržište softvera je veoma promenljivo i korisnički apetiti su sve veći, stoga je softverskim kompanijama od krucijalne važnosti da brzo razviju proizvod kako bi uspeli da ga prodaju i zadrže klijente.

Jedno od rešenja ovog problema jeste modelom upravljani razvoj softvera (Model Driven Software Development – MDSD). Modelom upravljani razvoj softvera predstavlja paradigmu koja se fokusira na modele kao primarni koncept. Sa stanovišta MDSD, modelovanje je uspešno ukoliko modeli imaju smisla iz ugla korisnika koji je upoznat sa domenom i ako mogu poslužiti kao osnova za implementaciju sistema. Kreiranje modela može biti naporno u situacijama gde se iziskuje obimna komunikacija među menadžerima proizvoda, dizajnerima, programerima i korisnicima domena aplikacija.

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

Rezultati primene MDSD paradigme imaju veliki broj prednosti, neke od njih su [2]:

- Brže vreme razvoja složenih poslovnih sistema, samim tim i smanjeni troškovi
- Softver je manje sklon greškama, što dovodi do povećanja kvaliteta
- Softver je adaptivniji, lakše se prilagođava novim korisničkim zahtevima, kao i novim tehnologijama
- Ostavlja prostora programerima da se bave kreativnijim stvarima, moguće je više se posvetiti poslovnoj problematici
- Premošćuje jaz između tržišta i IT sektora
- Pomaže da se arhitektura maksimalno iskoristi
- Sami modeli predstavljaju izvor znanja o domenu, uz to je i lakše dokumentovanje.

Softverska proizvodna linija (Software product line – SPL) se bavi sistematski podržanim ponovnim korišćenjem sredstava za razvoj aplikativnih domena. Sličan je masovnoj proizvodnji u tradicionalnoj industriji, sa ciljem da razvije softverske sisteme kao kvalitetne proizvode, uz smanjenje napora i vremena razvoja neophodnog da proizvod stigne do tržišta. Planirana i sistematska ponovna upotreba olakšana je u okviru porodice sistema objedinjenjem zajedničkih i varijabilnih aspekata u artefakte za višekratnu upotrebu [3]. Integracija ovih artefakata najčešće je omogućena tako što bazične softverske komponente imaju adaptivnu arhitekturu. Međutim, i pored svih olakšica, samo sastavljanje predstavlja najčešće ponavljajući posao.

Jezici specifični za domen (Domen Specific Language - DSL) su jezici koji su posebno dizajnirani za određeni, usko definisan domen, kontekst ili kompaniju, da olakšaju zadatak ljudi da opišu stvari u tom domenu. Ako je jezik usmeren na modeliranje, on se takođe može nazvati i jezik za modeliranje specifičan za domen (DSML) [4].

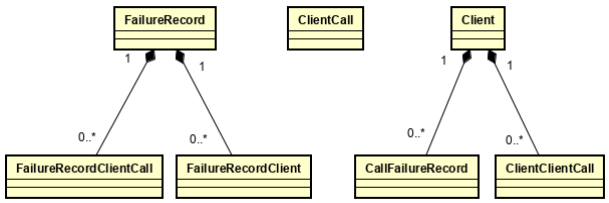
2. DOMEN REŠAVANOG PROBLEMA

Domen rešavanog problema je upotreba SPL radnog okvira za razvoj softvera za upravljanje sistemima javnog snabdevanja, sa akcentom na radnim nalozima i pratećim dokumentima koji se kreiraju pri ispadima sistema.

Upravljanje radnim nalozima za obradu ispada koje prijavljuju korisnici, predstavlja jednu od osnovnih funkcionalnosti poslovnog softvera koji se bavi upravljanjem sistemom javnog snabdevanja. Dokumenti koji se kreiraju tom prilikom su: ClientCall (prijava kvara od strane potrošača) i FailureRecord (dokument kojim se

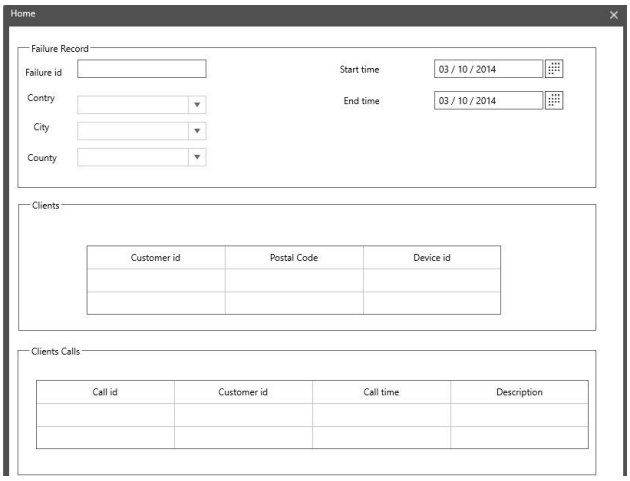
aktivira otklanjanje kvara). Client je dokument koji sadrži podatke o potrošaču (Slika 1). Sadržaj ovih dokumenata je specifičan za svaki sistem javnog snabdevanja. Pošto svaki takav sistem ima ustaljene poslovne procese, neophodno je da im se, prilikom kupovine softvera, ti dokumenti prilagode. Ovakvo prilagođavanje softvera pojedinačnim klijentima predstavlja dobar primer upotrebe SPL radnog okvira, kod kojeg bi se bazični softver proširio i iskonfigurisao s ciljem dobijanja proizvoda u skladu sa zahtevima. Proširenje ili kreiranje dokumenata podrazumeva sledeće aktivnosti:

1. proširenje modela podataka i konfiguracionih datoteka, kako bi sistem mogao da rukuje unetim podacima
2. proširenje šeme baze podataka, kako bi se dokument mogao trajno arhivirati
3. proširenje korisničkog interfejsa, da bi se korisniku omogućilo ažuriranje i pregled dokumenata
4. proširenje validacije unetih dokumenata i izmenu njihovog životnog ciklusa.



Slika 1 Zavisnost među dokumentima

Proširenja su dosta kompleksna, budući da su dokumenti međusobno zavisni i da se na korisničkom interfejsu jednog dokumenta vide određena polja drugog dokumenta (Slika 2). Izgled korisničkog interfejsa je u većini slučajeva tipiziran, ali nekada naručilac softvera zahteva specifičan izgled i ponašanje. Stoga, stopostotno generisanje koda nije moguće, pa je potrebno obezbediti podršku i za ručne izmene od strane razvojnog tima, uz njihovo očuvanje pri sledećem generisanju koda.



Slika 2 Ekranska forma Failure Record dokumenta

3. IMPLEMENTACIJA REŠENJA

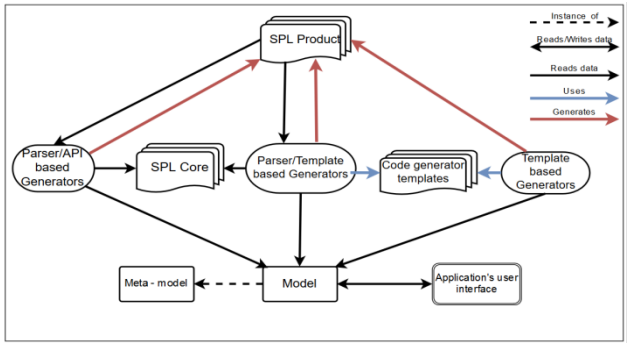
3.1 Osnovni zahtevi

Osnovni zahtev koji se postavlja pred generator je da se njime može generisati potpuno nov projekat, nastaviti razvoj već započetog projekta, kao i obezbediti podrška za rad različitih razvojnih timova koji održavaju određene tehnološke segmente (web, desktop, relacionu bazu i sl.). Neophodno je dozvoliti korisniku da proširi određenu grupu funkcionalnosti, odnosno da grafički korisnički interfejs bude intuitivan i jednostavan za korisnika koji ima domensko znanje. Na primer, ako korisnik želi da proširi isključivo desktop aplikaciju i bazu, generator treba da podrži samo ova proširenja.

Nakon generisanja, neophodno je da se sve datoteke integrišu sa već postojećim projektom. Dakle, neophodno ih je registrovati u .NET projektu (.csproj i .sqlproj datoteka), kao i dodati neophodne reference na biblioteke. Željeni rezultat dodavanja je prošireni sistem koji se uspešno kompajlira i izvršava.

3.2 Arhitektura generatora

U zavisnosti od vrste datoteke, postoje različiti načini generisanja koda. Budući da su oni obrađeni u [5] i nisu tema ovog rada, biće samo pobrojani. Njihov prikaz je dat na slici (Slika 3). Prva grupa generatora sastoji se od generatora koji generišu kod isključivo pomoću parsera (manipulacijom nad sintaksnim stablima). Drugu grupu generatora čine ranije navedeni generatori koji svaki put generišu datoteku isključivo korišćenjem šablona (što je redak slučaj). Najčešće upotrebljavani slučaj je upravo kombinacija prethodna dva tipa. Generator prvo proveri da li datoteka postoji, pa ako ne postoji, generiše se putem šablona, a ako postoji, kod se generiše manipulacijom nad sintaksnim stablima.

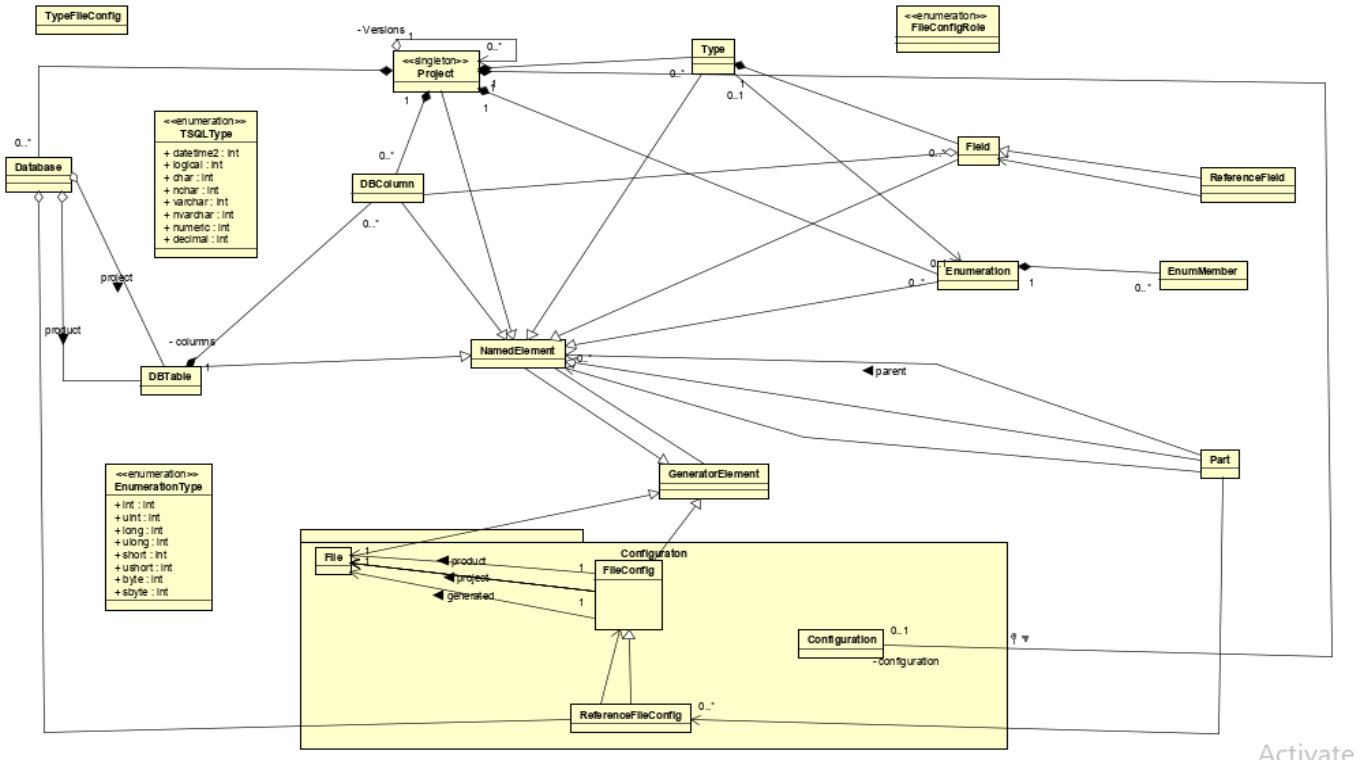


Slika 3 Arhitektura generatora

3.3 Meta-model

Meta-model jezika za specifikaciju proširenja se sastoji od nekoliko celina koje su prikazane na slici 4. Celine su:

- Project
- Konfiguracija
- Dokument i njegovi elementi
- Baza podataka



Slika 4 Meta-model jezika za opis uzajamno zavisnih dokumenata

Project predstavlja centralnu komponentu meta-modela, to jest root element. On sadrži sve ostale elemente. Kako bi se pratilo prethodno stanje projekta i mogao lakše razlikovati ručno pisani kod od generisanog, projekat sadrži kolekciju verzija prethodnih modela. Jedinstveni identifikator verzije je *versionId*.

Project sadrži kolekciju *Configuration* meta-klasa. Atributi ove meta-klase su:

1. *projectFileConfigs* - unutar ove kolekcije se nalazi konfiguracija za datoteke koje su nezavisne od konkretnog tipa dokumenta (obično postoji jedna takva datoteka na nivou celog projekta). Primera radi, ovde spadaju resx datoteke koji se koriste za lokalizaciju, zatim razni konfiguracioni fajlovi za servise i tako dalje.

2. *typeFileConfigs* - unutar ove kolekcije se nalazi TypeFileConfig konfiguracija za datoteke koje su vezane za sam dokument.

3. *templateFileConfigs* - unutar ove kolekcije se nalazi konfiguracija za datoteke koje su vezane za novokreirane dokumente. Primer bi bile datoteke koje se koriste za šablon MVVM (Model – View – ViewModel), zatim datoteke za web klijenta, gde se koristi MVC (Model – View – Controler) šablon i drugi.

Meta-klasa *Type* opisuje dokument. Ona nasleđuje *NamedElement*. Kolekcija ovog tipa nalazi se u meta-klasi *Project*. Sadrži kolekciju tipa *Field*.

Meta-klasa *Field* opisuje polje unutar dokumenta. Nasleđuje *NamedElement*. Field sadrži i informaciju o tipu podatka. Pošto se radi o .NET tehnologijama, korišćena je enumeracija *CSharpType* koja specificira moguće tipove polja. Takođe, *Field* sadrži atribut kojim se definiše da li treba ili ne da se njegova vrednost arhivira u relacionoj bazi podataka, tj. da li za njega treba da se odradi objektno-relaciono mapiranje.

Ključni element meta-modela predstavlja meta-klasa *Part*. Ona omogućava dodavanje koda u specificirane delove datoteke. Delovi mogu biti različite metode dokumenata, pozivi metoda, različiti parametri u zavisnosti od referenciranog polja i tako dalje. Kolekcija ovog elementa sadržana je u meta-klasi *Type*.

Meta-klasa *Database* opisuje pojavu tipa entiteta baze podataka u stvarnom poslovnom sistemu. Ona nasleđuje *NamedElement*, dakle, sadrži jedinstveni identifikator i ime. Pored toga, sadrži listu *ReferenceFileConfig*-a, upravo zbog toga što i fajlovi kojima se proširuju baze mogu biti deljeni. Instanca meta-klase *Project* sadrži kolekciju objekata *Database*.

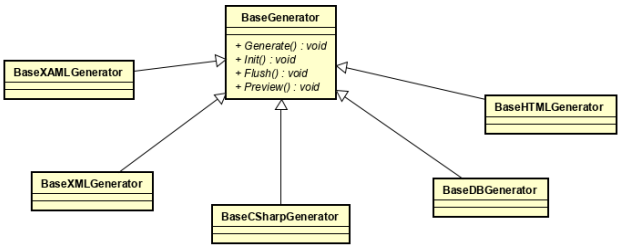
Kako je reč o SPL-u, baza podataka ima dve kolekcije tabela. Jedna se odnosi na osnovni skup tabela iz jezgra SPL-a i naziva se *product*, dok se druga kolekcija tebela odnosi na skup tabela proširenja i naziva se *project*. Pored baze podataka moguće je modelovati tabele i kolone. Potrebno je naglasiti da baza ne sadrži meta-klasu za modelovanje primarnog ključa, pošto dato rešenje ne dozvoljava postojanje kompozitnih ključeva.

3.4 Hijerarhija generatorskih klasa

Kako bi se objedinile zajedničke funkcionalnosti generatora, u cilju umanjenja redudanse koda i definisanja strukture koju bi implementacije konkretnih generatora trebalo da imaju, osmišljena je hijerarhija baznih generatora. Hijerarhija je vezana za tipove datoteka kojima CG rukuje (Slika 5).

CG sadrži sledeće bazne generatore:

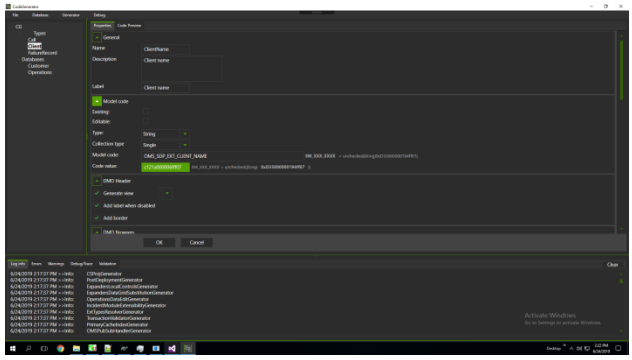
1. BaseXAMLGenerator
2. BaseXMLGenerator
3. BaseCSharpGenerator
4. BaseDBGenerator
5. BaseHTMLGenerator



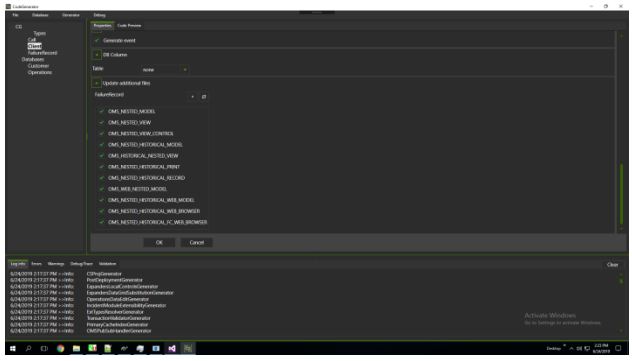
Slika 5 Hijerarhija baznih generatora

3.5 Grafički korisnički interfejs

Kako bi specifikacija modela bila olakšana, za razvijeni jezik je kreirano radno okruženje pomoću kog se zadaje model koji je ulaz za generator koda. Grafički korisnički interfejs je prikazan na slikama 6 i 7.



Slika 5 Unos novog polja u dokument



Slika 6 Selekcija delova dokumenata unutar referencirajućeg dokumenta

4. ZAKLJUČAK

Ovim radom predstavljena je arhitektura generatora koda čija je svrha automatizacija razvoja uzajamno zavisnih dokumenata u okviru softverskih proizvodnih linija. Opisani su problemi postavljeni pred generator koji su specifični za datu arhitekturu sistema. Jedan od glavnih izazova bila je potreba da generator podrži projekte u različitim fazama razvoja, zbog čega je bilo neophodno koristiti razne vrste parsera, pored klasičnog generisanja koda korišćenjem obrađivača šablona.

U radu je prikazan meta-model jezika, globalna arhitektura generatora i korisnički interfejs generatorske aplikacije. Meta-model je izložen detaljno, uz navođenje primera proširenja sistema. Grafički korisnički interfejs je predstavljen kroz primere proširenja uzajamno zavisnih dokumenata.

Kada je reč o daljem pravcu razvoja generatorske aplikacije, planirana je implementacija proširenja web klijenta, budući da implementacija ovog dela sistema nije završena. Takođe se kao jedan od dodatnih zahteva nametnulo i uređivanje redosleda polja kod deljenih delova dokumenta.

5. LITERATURA

[1] Shapiro RJ. The us software industry as an engine for economic growth and employment. Georgetown McDonough School of Business Research Paper. 2014 Sep 22(2541673).

[2] Haan, J. Dean. "15 reasons why you should start using Model Driven Development." (2009).

[3] Urli, Simon, Mireille Blay-Fornarino, and Philippe Collet. "Handling complex configurations in software product lines: a tooled approach." *Proceedings of the 18th International Software Product Line Conference-Volume 1*. ACM, 2014.

[4]Brambilla, Marco, Jordi Cabot, and Manuel Wimmer. "Model-driven software engineering in practice." *Synthesis Lectures on Software Engineering* 3.1 (2017): 1-207.

[5] Nenad Todorović.”Arhitektura generatora koda za softverske proizvodne linije”. Master rad. Fakultet tehničkih nauka. 2017.

Biografija:

Tijana Lalošević je rođena 30. avgusta 1994. u Novom Sadu. Završila je osnovnu školu „Dušan Radović“ u Novom Sadu, kao dobitnik Vukove diplome. Nakon osnovne škole je završila gimnaziju „Svetozar Marković“ u Novom Sadu, takođe kao dobitnik Vukove diplome. Školske 2014/2015. godine upisala je Fakultet Tehničkih Nauka u Novom Sadu, odsek Računarstvo i automatika, usmerenje Primenjene računarske nauke i informatika. Na četvrtoj godini studiranja je prešla na odsek Elektroenergetski softverski inženjering. Diplomirala je 2018. godine sa prosečnom ocenom 9,95. Nakon osnovnih studija upisala je master studije na smeru Informacioni inženjering. Položila je sve ispite propisane planom i programom sa prosečnom ocenom 10.00. Dvostruki je dobitnik Dositejeve nagrade.