Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/03BE11Keselj

INTERPRETER REST API SERVISA BAZIRAN NA OPENAPI SPECIFIKACIJI

REST API SERVICES INTERPRETER BASED ON OPENAPI SPECIFICATION

Sergej Kešelj, *Fakultet tehničkih nauka, Novi Sad*

Oblast- ELEKTROTEHNIKA I RAČUNARSTVO

Kratak sadržaj – *Ovaj rad prikazuje jednu implementaciju softverskog sistema za interpretiranje modela i njegovo izvršavanje u obliku generičkih REST API servisa opisanih OpenAPI dokumentom.*

Ključne reči: *MDE, MDA, REST, OpenAPI*

Abstract – *This paper features an implementation of a software system for interpreting and execution of models as generic REST API services described by an OpenAPI document.*

Keywords: *MDE, MDA, REST, OpenAPI*

1. UVOD

Od prve pojave veb interfejsa za programiranje aplikacija, novine poput računarstva u oblaku ili platformi prenosivih uređaja, motivisale su promene i u pristupu razvoju interfejsa uslovljenih ovim novim potrebama.

Potreba za standardizacijom protokola i pristupa razvoju uslovila je pojavu REST stila arhitekture softverskih sistema 2000. godine. Mrežni interfejsi za programiranje koji su u skladu sa ograničenjima REST stila arhitekture se nazivaju i RESTful interfejsima. Ovaj stil arhitekture se snažno oslanja na postojeću infrastrukturu interneta, kao što je HTTP (*Hypertext Transfer Protocol*), URL (*Uniform Resource Locator*), internet vrste medija ili HTTP glagoli.

Tehnike brzog razvoja softvera poput MDE (Model Driven Engineering) ili agilnih metodologija razvoja, mogu nam teorijski i praktično pomoći u brzom razvoju složenih softverskih sistema. Neke od tehnika brzog razvoja softvera uključuju i generisanje izvršnog koda rešenja koji se dalje, zajedno sa ručno pisanim prilagođenim kodom isporučuje u produkciona okruženja.

Ovaj rad se bavi implementacijom rešenja za dinamičko izvršavanje modela, korišćenjem tehnika inženjerstva vođenog modelima.

Ovako interpretirane modele, pomoću generičkih RESTful interfejsa za programiranje aplikacija, možemo opisati dinamički generisanim standardizovanim OpenAPI dokumentima (slika 1). Ovo će nam omogućiti da koristimo postojeća rešenja za generisanje klijentskih aplikacija za ovako interpretirane RESTful interfejse, jer ćemo kao ulaznu informaciju moći da koristimo generisan OpenAPI dokument koji opisuje naš interpretirani model, odnosno njegov generički RESTful interfejs (slika 2).

NAPOMENA:

Ovaj rad proistekao je iz master rada čiji mentor je bila dr Gordana Milosavljević, vanr. prof.

2. RAZVOJ SOFTVERA VOĐEN MODELIMA

Razmatraćemo razvoj softvera vođen modelima kroz sledeće bitne elemente:

1. Inženjerstvo upravljano modelima
2. Arhitekture upravljane modelima
3. Generisanje koda iz modela
4. Izvršavanje, odnosno interpretiranje modela
5. Evoluciju modela.

2.1. Inženjerstvo upravljano modelima

Apstrakcije su jedan od osnovnih principa inženjeringa softvera. Apstrakcije se iz realnog okruženja prevode u računarsko okruženje pomoću modela. Modelovanje i transformacije modela čine srž razvoja upravljanog modelima. Modele je moguće rafinisati i transformisati u konkretne tehničke implementacije, odnosno softverske sisteme.

OMG (*Object Management Group*) konzorcijum za tehnološke standarde opisuje inženjerstvo upravljano modelima kao metodologiju razvoja softvera koja se fokusira na kreiranje i korišćenje domenskih modela, koji predstavljaju konceptualne modele za sve teme vezane za specifičan problem koji se opisuje.

Moglo bi se reći da MDE (*Model-driven Engineering*) ima za cilj uspostavljanje apstraktne reprezentacije znanja i aktivnosti u okviru određenog aplikativnog domena. Ovo je drugačiji pogled na problematiku implementacije softvera koja se oslanja na pristupe okrenute algoritamskim konceptima.

Smatra se da je MDE paradigma uspešna ukoliko proizvedeni modeli imaju smisla za poznavaoce domena, dok istovremeno mogu poslužiti kao osnova za automatizovanu implementaciju sistema. Cilj je da se modeli razvijaju kroz intenzivnu saradnju i komunikaciju između svih strana umešanih u projekat.

Kako je jedan od ciljeva OMG konzorcijuma da svaki standard ili metodologija ima i konkretne alate i principe primene, nastao je pristup razvoju softverskih sistema pod nazivom MDA (*Model-driven architecture*).

Slika1. Proces interpretacije modela na serveru

Slika 2. Proces generisanja klijenta

2.2. Arhitektura upravljana modelima

Prvi put 2000. godine OMG konzorcijum pominje MDA (*Model-driven architecture*) kao viziju razvoja softvera koja se oslanja na vezivanje modela u celinu koja bi predstavljala kompletan opis sistema. Ovaj pristup bi trebalo da koristi postojeće tehnologije koje podržavaju postojeće, ali i buduće OMG standarde i kao rezultat toga daju podršku razvoju upravljanom modelima, tako da modeli postanu dobra koja, priliko promene tehnološke platforme, mogu da se ponovo iskoriste, umesto troška implementacije svakog pojedinačnog sistema od nule.

Danas modeli i dalje najčešće predstavljaju trošak. Jednom kada je model dizajniran, mora biti transformisan u izvršni kod. Ovo je težak, dugotrajan, sklon greškama i pre svega skup proces. Dalje, nakon što je posao apstrakovanja realnog domena u modele završen, samo je transformacija iz izvornog koda u izvršni kod automatizovana.

MDA je rezultat prepoznavanja da je modelovanje, interoperabilnost i nezavisnost od konkretnih tehničkih platformi dobra stvar. Ovaj pristup dozvoljava programerima da izgrade modele bez znanja o drugim modelima koji će činiti krajnji sistem. Ovakvi modeli se kombinuju u konačni sistem tek na kraju procesa, odnosno što kasnije u procesu. Ovo sprečava odluke koje su vezane za dizajn rano u procesu, pa samim tim čini modele nezavisnim od platforme za implementaciju krajnjeg sistema. U krajnjem odredištu, aplikacija postaje nezavisna od implementacije, pa može biti kombinovana sa mnogim tehnologijama, kao i drugim aplikacijama i sistemima. Moglo bi se govoriti o nekoj vrsti interoperabilnosti u dizajnu, što naše modele čini vrednostima, a ne troškovima.

2.3. Generisanje koda iz modela

Generisanje koda je dobro poznati način izvođenja implementacije iz modela sistema. Računar izvodi izvršni kod direktno iz modela. Ovaj proces može rezultirati različitim kvalitetom generisanog koda. Često se proizvodi samo osnovni kod. Ovaj osnovi kod čine nekompletni fragmenti koda sa namerom da isti budu kompletirani od strane programera u jeziku generisanog koda.

Ukoliko modeli sadrže kompletne informacije o sistemu moguće je generisati kompletno operativan sistem iz modela. U ovom slučaju generisani kod može da se kompajlira u izvršni sistem bez potrebe za intervencijom programera. Tada, kod najčešće niti može, niti bi trebalo da bude modifikovan od strane čoveka, jer bi to prouzrokovalo problem sa integracijom izmena nazad u modele. Samo je sistem kompletno generisan iz modela u potpunoj saglasnosti sa samim modelom. Kako je očekivano da modeli budu u ekvivalenciji sa sistemom, nije poželjno dozvoliti izmene u generisanom kodu. Ovakve izmene bi morale biti načinjene i u modelima. Iz tog razloga samo su izmene na modelima poželjne.

Bez obzira na cilj da se izmene rade samo u modelima, većina alata koja generiše kod nije u stanju da generiše kompletan sistem. Često je neophodno modifikovati generisani kod, što implicira da modeli nemaju kompletnu informaciju o sistemu koji je modeliran. Ovo se naročito odnosi na MDA nivo modelovanja sistema.

2.4. Interpretiranje modela

Jedan način sinhronizacije implementacije i dizajna jeste korišćenje modela kao implementacije. Jezik za opis modela bi morao biti dovoljno informativan da pruži dovoljno detalja za implementaciju.

Druga komponenta ovakvog sistema bi bila neka vrsta virtualne mašine koja bi bila u stanju da izvršava ovako opisane modele. Korišćenjem virtualne mašine eliminišemo potrebu da se model transformiše u različite programske jezike korišćenjem generatora koda.

Ovakav pristup pruža nekoliko prednosti. Model postaje direktno izvršiv na svakoj platformi koja obezbeđuje odgovarajuću virtualnu mašinu. Ovo eliminiše dodatni napor potreban da bi se generisao novi izvršni kod svaki put kada se menja ciljana platforma ili kada se dodaje neki novi mehanizam u softversko rešenje koje se već izvršava na ciljanoj platformi.

Ovakva ažuriranja se mogu vršiti direktno na virtuelnoj mašini bez potrebe da se menja model koji se izvršava. Dakle, za razliku od pristupa sa generisanim kodom, ne postoji potreba za ponovnim generisanjem za novu platformu ili za novu verziju postojeće platforme.

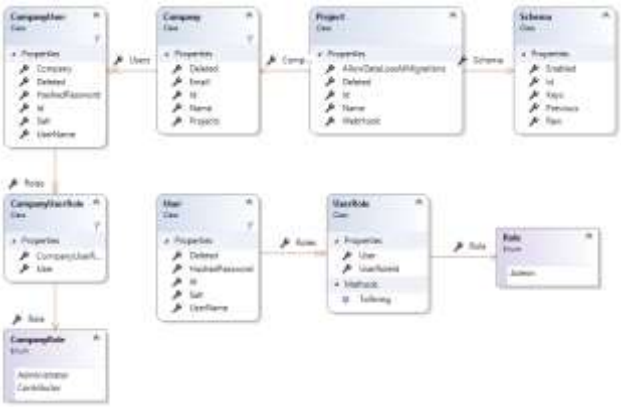
Ovo je značajno unapređenje, jer sve izmene koje bi uvodili u virtuelno okruženje mogu da se primene na sve postojeće sisteme, prethodno modelirane, a koji se već nalaze u izvršnom stanju. Ovo višestruko smanjuje vreme u toku razvoja. Može se smatrati da je potrošeno vreme minimalno moguće.

2.4. Evolucija modela

Samo jedna stvar je konstantna u razvoju softverskih sistema, a to je da će biti promena. Ova jednostavna činjenica je uticala na velike izmene u pristupu razvoju softvera i između ostalog dovela nas od *waterfall* modela do agilnog manifesta, pa dalje do agilnih metodologija razvoja softvera.
U skladu sa ovim trendom, može se govoriti o evolutivnom razvoju softvera i svih njegovih komponenti, odnosno slojeva. U kontekstu MDE možemo govoriti o agilnom pristupu evoluciji baza podataka kao najčešćeg krajnjeg odredišta podataka većine softverskih sistema. Konkretno pitanje koje se postavlja je kako možemo automatizovati detekciju i primenu izmena modela na postojeće softverske sisteme u svetlu interpretacije modela?
Postoji ograničen skup operacija koje možemo primeniti kako bismo automatizovali razumevanje izmena nad modelom i njihovog uticaja na model baze podataka. Izdvajaju se dva podskupa potencijalnih migracionih promena zavisno od toga da li dolazi do gubitka podataka evolucijom modela.

3. IMPLEMENTACIJA SISTEMA

Projekat je realizovan kao SaaS (Software as a Service) rešenje. Svaka kompanija može da upravlja svojim korisnicima i projektima. Projekat je predstavljen podacima o samom projektu, konfiguracijom projekta i modelom. Model je reprezentovan korišćenjem OpenAPI specifikacije šema objekta i OpenAPI proširenjem kojim se opisuju primarni ključevi modela (slika 3).



Slika 3. Prikaz dijagrama klasa sistema

3.1. Dinamička interpretacija modela

Modeli se u sistemu čuvaju u okviru entiteta *Schema*. Ovaj entitet čuva opis modela u polju *Raw*. Model se opisuje JSON ili YAML dokumentom po OpenAPI specifikaciji za opis OpenAPI info objekta. Primer 1 prikazuje jedan model u polju *Raw*:

```
Pet:
  required:
    - id
    - name
  properties:
    id:
      type: integer
      format: int64
    name:
      type: string
    family:
      type: string
```

Primer 1. Opis modela Pet

Zbog kompletnosti modela bilo je neophodno proširiti OpenApi specifikaciju meta podacima koji opisuju primarne ključeve modela. Za proširenje specifikacije iskorišćeno je svojstvo proširivosti OpenAPI specifikacije. Primer 2 daje primer proširenja specifikacije koja definiše meta podatke o primarnom ključu entiteta definisanog modelom:

```
x-entity-extensions:
  Pet:
    keys:
      - id
```

Primer 2. Opis proširenja meta-modela

3.2. Dinamičko generisanje tipova klasa

Na osnovu stabla parsiranja generišemo .NET klase u vreme izvršavanja aplikacije (slika 4). Ovako generisane klase možemo iskoristiti da uz pomoć objektno-relacionog mapera kreiramo bazu podataka koja bi odgovarala modelu. Ovaj korak je neophodan kako bismo mogli interpretirati modele i omogućiti njihovo izvršavanje u okviru našeg projekta, a bez međukoraka generisanja koda.

```
public static EntityTypes CompileResultTypes(this OpenApiDocument document,
                                             string company,
                                               string project)
{
    var result = new ConcurrentDictionary<string, EntityTypeMetadata>();
    var schemasWithPrimaryKey = document.Components.Schemas.AsEnumerable();
    var typeBuilders = new Dictionary<string, TypeBuilder>();
    var moduleBuilder = GetModuleBuilder(company, project);
    var assemblyName = $"RestApiInterpreter.WebApi.Tenants.{company}.Projects.{project}";

    foreach (var schema in schemasWithPrimaryKey)...

    foreach(KeyValuePair<string,TypeBuilder> tb in typeBuilders)
    {
        Type objectType = tb.Value.CreateType();
        result.TryAdd(objectType.Name, new EntityTypeMetadata() {
            Type = objectType,
            Keys = FindKeysFromExtensions(document, tb.Key),
            Required = FindRequiredFromExtensions(document, tb.Key) });
    }
    return new EntityTypes(result, company, project);
}
```

Slika 4. Prikaz koda za dinamičko kreiranje klasa

3.3. Dinamičko generisanje DBContext objekta

Nakon što smo dinamički kreirali klase možemo preći na perzistenciju objekata tako što ćemo dinamički kreirati DBContext objekat. Dinamičko ponašanje smo implementirali tako što smo nasledili DBContext klasu iz EntityFramework biblioteke i izmenili njeno ponašanje tako da šemu baze podataka kreira na osnovu klasa koje smo dinamički izgenerisali i dodali u kontekst. Kako se radi o multi-tenant rešenju, morali smo obratiti pažnju i na to da će svaki projekat imati različitu bazu i različit model baze.

3.4. Migracije šeme baze podataka

Evolucija modela direktno utiče na šemu baze podataka. Kako je evolucija produkcionih softverskih sistema neminovna, iskoristićemo EntityFramework podršku za migracije i omogućiti, zavisno od konfiguracije projekta, migracije sa i bez gubitka podataka.

Ponašanje implementiramo nasleđivanjem i proširivanjem ponašanja klase DbMigrationsConfiguration iz biblioteke EntityFramework.

3.5. Dinamička interpretacija REST API servisa

Potreban nam je generički ASP.NET Web API kontroler koji će prihvatati zahteve klijentskih aplikacija i izvršavati osnovne operacije nad interpretiranim modelima. Parametri akcije nam pomažu da odredimo tačan model koji želimo da izvršimo, a HTTP glagoli određuje vrstu operacije.

3.6. Generisanje OpenAPI dokumenta iz modela

Pošto imamo sve neophodne informacije o modelu i REST API servisu možemo iskoristiti OpenAPI .NET SDK da generišemo OpenAPI dokument koji opisuje REST API servis preko kog pristupamo izvršavanju modela. Ovakav dokument se može koristiti kao ulaz za druge alate, koji mogu generisati dokumentaciju REST API servisa ili klijentske aplikacije.

3.7. Generisanje klijentskih aplikacija

OpenAPI dokument koji opisuje REST API servis sadrži sve podatke potrebne za generisanje dokumentacije ili klijentskih aplikacija. Pošto je OpenAPI standardizovan, već postoji veći broj alata.

Mi ćemo ovde navesti primer korišćenja alata AutoRest. Ovaj alat može da se koristi sa različitih platformi (Windows, MacOS ili Linux) i da se pomoću njega generiše klijentski kod u nekom od ponuđenih programskih jezika poput: C#, Go, Java, Python, NodeJS i drugih.

Kako je alat proširiv i otvorenog koda, za očekivati je da se podrška za jezike i dalje unapređuje.

6. ZAKLJUČAK

Inženjerstvo vođeno modelima i arhitekture upravljane modelima gledaju na problematiku implementacije softverskih sistema kao dobro definisanih modela koji su nezavisni i samodovoljni. Ovako definisani modeli postaju vrednost koja može da se upotrebljava za građenje konačnih softverskih sistema, kao i za ponovnu upotrebu u različitim softverskim sistemima.

Arhitektura upravljana modelima daje dva pristupa problemu transformacije modela u konačni izvršni softverski sistem: pomoću generisanja koda ili pomoću softverskog sistema za interpretaciju modela.

U ovom radu smo se bavili implementacijom softverskog sistema koji bi bio u stanju da izvršava modele tako što bi generisao sve potrebne elemente dinamički u vreme izvršavanja i praktično interpretirao modele bilo kog sistema opisanog modelima.

Rešili smo da sistem implementiramo kao generički REST API koji će biti u stanju da generiše OpenAPI dokument koji opisuje REST API posmatranog modela.

Ovako generisan dokument smo koristili kao ulaz za alate koji na osnovu OpenAPI dokumenta mogu da generišu dokumentaciju ili klijentske aplikacije, poput AutoRest alata.

Tako smo uspešno zaokružili sistem koji omogućava korisniku kreiranje i interpretaciju, odnosno izvršavanje modela, bez potrebe za generisanjem izvornog koda.

Dodatno, ovaj softverski sistem omogućava evoluciju modela pomoću migracija podataka, kao i povezivanje sa drugim sistemima mehanizmima definisanja *WebHook* URL adresa.

Ovaj sistem predstavlja dokaz koncepta i kao takav ima dovoljno prostora za dalja unapređenja. Potrebno je implementirati sigurnosne mehanizme u opis modela. Ovo bi se moglo postići korišćenjem nekih od sigurnosnih mehanizama opisanih OpenAPI specifikacijom, a implementiranih u tehnologijama koje koristimo u ovom projektu. Podrška za evoluciju modela bi mogla da se poboljša uvođenjem bolje kontrole operacija koje se vrše u toku migracija. Još jedan od pravaca unapređenja sistema bi mogao biti bolja kontrola instrumentacije sistema uvođenjem dnevnika aktivnosti koji bi mogao da se konfiguriše na nivou projekta.

7. LITERATURA

[1] L. Richardson, S. Ruby (2007), RESTful Web Services, O'Reilly Media, Inc. Sebastopol, CA

[2] Kent S. (2002) Model Driven Engineering. In: Butler M., Petre L., Sere K. (eds) Integrated Formal Methods. IFM 2002. Lecture Notes in Computer Science, vol 2335. Springer, Berlin, Heidelberg

[3] Beydeda S., Book M., Volker G. (2005) Model-Driven Software Development, Springer-Verlag GmbH ISBN 978-3-540-28554-0

[4] Stephen J. Mellor, Kendall Scott, Axel Uhl, Dirk Weise (2004) MDA Distilled: Principles of Model-driven Architecture

[5] Tim Schattkowsky (2003), Model-based Development of Embedded Systems: Executable Models vs. Code Generation

[6] Scott W. Ambler, Agile Database Techniques— Effective Strategies for the Agile Software Developer

[7] F. Chong, G. Carraro, R. Wolter (2006), Multi-Tenant Data Architecture, Microsoft Corporation

Kratka biografija:

Sergej Kešelj rođen je u Mostaru 1979. god. Osnovne studije na Prirodno-matematičkom fakultetu u Novom Sadu, iz oblasti Informatike je završio 2006. godine. Od 2016. je student akademskih master studija na Fakultetu tehničkih nauka na smeru Softversko inženjerstvo. Master rad je odbranio 2018. godine.