



УНИВЕРЗИТЕТ У НОВОМ САДУ  
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У  
НОВОМ САДУ



Небојша Кнежевић, ПР 104/2015

**Испитивање могућности Јена РДФ складишта за  
рад са СИМ подацима**

**ДИПЛОМСКИ РАД**  
**- Примењено софтверско инжењерство (ОАС) -**

Нови Сад, 2023



УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА  
21000 НОВИ САД, Трг Доситеја Обрадовића 6


## КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

|  |   |
|--|---|
| Редни број, <b>РБР</b> :   |   |
| Идентификациони број, <b>ИБР</b> :   |   |
| Тип документације, <b>ТД</b> :   | Монографска публикација   |
| Тип записа, <b>ТЗ</b> :  | Текстуални штампани документ/ ЦД  |
| Врста рада, <b>ВР</b> :  | Дипломски рад   |
| Аутор, <b>АУ</b> :   | Небојша Кнежевић  |
| Ментор, <b>МН</b> :  | др Милан Гаврић, доцент   |
| Наслов рада, <b>НР</b> :   | Испитивање могућности Јена РДФ складишта за рад са СИМ подацима   |
| Језик публикације, <b>ЈП</b> :   | Српски (латиница)   |
| Језик извода, <b>ЈИ</b> :  | Српски/енглески   |
| Земља публикавања, <b>ЗП</b> :   | Србија  |
| Уже географско подручје, <b>УГП</b> :  | Војводина   |
| Година, <b>ГО</b> :  | 2023  |
| Издавач, <b>ИЗ</b> :   | Ауторски репринт  |
| Место и адреса, <b>МА</b> :  | Факултет техничких наука (ФТН), Д. Обрадовића 6, 21000 Нови Сад   |
| Физички опис рада, <b>ФО</b> :<br>(поглавља/страна/ цитата/табела/слика/графика/прилога) | 5/43/11/0/43/0/0  |
| Научна област, <b>НО</b> :   | X   |
| Научна дисциплина, <b>НД</b> :   | X   |
| Предметна одредница/Кључне речи, <b>ПО</b> :   | <b>RDF, CIM, JENA, FUSEKI</b> , моделовање, парсирање, тестирање, испитивање  |
| <b>УДК</b>   |   |
| Чува се, <b>ЧУ</b> :   | Библиотека ФТН, Д. Обрадовића 6, 21000 Нови Сад   |
| Важна напомена, <b>ВН</b> :  |   |
| Извод, <b>ИЗ</b> :   | Рад описује начин моделовања дистрибутивног извода уз помоћ различитих софтверских алата, програмирање корисничке апликације која омогућује парсирање овог модела, затим складиштење модела у оквиру РДФ складишта и извршавање одређених операција и упита над таквим моделом. Сваки од ових корака је детаљно описан и наведени су коришћени алати и технологије са разложима њихове употребе. Коначно извршено је испитивање перформанси ЈЕНА РДФ складишта и релационих база података приликом њиховог тестирања фајловима различите величине и приложене су слике и коментари који интерпретирају добијене резултате у оквиру контекста тестирања и дају даље смернице за ефикасан рад са РДФ СИМ фајловима. |
| Датум прихватања теме, <b>ДП</b> :   |   |
| Датум одбране, <b>ДО</b> :   |   |
| Чланови комисије, <b>КО</b> :  | Председник: X   |
|  | Члан: X   |
|  | Члан, ментор: X   |
|  | Потпис ментора  |



## KEY WORDS DOCUMENTATION

|  |   |
|--|---|
| Accession number, <b>ANO</b> :   |   |
| Identification number, <b>INO</b> :  |   |
| Document type, <b>DT</b> :   | Monographic publication   |
| Type of record, <b>TR</b> :  | Textual material, printed/CD  |
| Contents code, <b>CC</b> :   | Bachelor thesis   |
| Author, <b>AU</b> :  | Nebojša Knežević  |
| Mentor, <b>MN</b> :  | dr Milan Gavrić, Full Professor   |
| Title, <b>TI</b> :   | Examining the capabilities of Jena RDF storage in aspect of working with CIM data   |
| Language of text, <b>LT</b> :  | Serbian   |
| Language of abstract, <b>LA</b> :  | Serbian/English   |
| Country of publication, <b>CP</b> :  | Serbia  |
| Locality of publication, <b>LP</b> :   | Vojvodina   |
| Publication year, <b>PY</b> :  | 2023  |
| Publisher, <b>PB</b> :   | Author's reprint  |
| Publication place, <b>PP</b> :   | Faculty of Technical Sciences, D. Obradovića 6, 21000 Novi Sad  |
| Physical description, <b>PD</b> :<br>(chapters/pages/ref./tables/pictures/graphs/appendixes) | 5/43/11/0/43/0/0  |
| Scientific field, <b>SF</b> :  | X   |
| Scientific discipline, <b>SD</b> :   | X   |
| Subject/Key words, <b>S/KW</b> :   | <b>RDF, CIM, JENA, FUSEKI</b> , modeling, parsing, testing, querying  |
| <b>UC</b>  |   |
| Holding data, <b>HD</b> :  | Library of Faculty of Technical Sciences, D. Obradovića 6, 21000 Novi Sad   |
| Note, <b>N</b> :   |   |
| Abstract, <b>AB</b> :  | The paper describes the way to model distributional feeder with a help of various software tools, programming the user application that enables the parsing of this model, then storing the model within RDF storage and executing certain operation and queries on such a model. Each of these steps is described in detail and reader is provided with a list of the tools and technologies used, alongside reasons for their use. Finally, results of the testing of Jena RDF storage and relational databases performance were provided by using a number of files of various size, those results are shown on number of pictures alongside attached commentary that interpret obtained results within the context of way of testing and provide further guidelines for efficient use when working with RDF CIM data. |
| Accepted by the Scientific Board on, <b>ASB</b> :  |   |
| Defended on, <b>DE</b> :   |   |
| Defended Board, <b>DB</b> :  | President: X  |
|  | Member: X   |
|  | Member, Mentor: X   |
|  | Mentor's sign   |

|   |  |               |
|---|--|---------------|
|  | УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА<br>21000 НОВИ САД, Трг Доситеја Обрадовића 6 | Датум:        |
|   | <b>ЗАДАТАК ЗА ИЗРАДУ ДИПЛОМСКОГ<br/>(BACHELOR) РАДА</b>  | Х             |
|   |  | Лист/Листова: |
|   |  | Х             |

(Податке уноси предметни наставник - ментор)

|                                  |   |
|----------------------------------|---|
| Врста студија:                   | <input checked="" type="checkbox"/> Основне академске студије<br><input type="checkbox"/> Основне струковне студије |
| Студијски програм:               | Примењено софтверско инжењерство  |
| Руководилац студијског програма: | Х   |

|          |                         |               |             |
|----------|-------------------------|---------------|-------------|
| Студент: | Небојша Кнежевић        | Број индекса: | ПР 104/2015 |
| Област:  | Х                       |               |             |
| Ментор:  | др Милан Гаврић, доцент |               |             |

НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ДИПЛОМСКИ (Bachelor) РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:

- проблем – тема рада;
- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;
- литература

### НАСЛОВ ДИПЛОМСКОГ (BACHELOR) РАДА:

Испитивање могућности Јена РДФ складишта за рад са CIM подацима

### ТЕКСТ ЗАДАТКА:

**Modelovanje distributivnog izvoda u CIM-u, a potom i definisanje CIM profila i CIM/XML fajlova koje je potrebno parsirati u okviru aplikacije, importovati u RDF skladište i definisati nekoliko SPARQL upita nad takvim modelom, a zatim i testirati mogućnosti i performanse Jena RDF skladišta prilikom rada sa različitim RDF fajlova i uporediti te performanse u odnosu na relacione baze podataka.**

|                                  |              |
|----------------------------------|--------------|
| Руководилац студијског програма: | Ментор рада: |
|                                  |              |

Примерак за: ☐ - Студента; ☐ - Ментора

# 1 SADRŽAJ

|          |  |           |
|----------|--|-----------|
| <b>2</b> | <b>OPIS REŠAVANOG PROBLEMA.....</b>  | <b>2</b>  |
| 2.1      | DEFINISANJE MODELA DISTRIBUTIVNOG IZVODA .....   | 2         |
| 2.2      | PROGRAMIRANJE APLIKACIJE I SPREGA SA SKLADIŠTEM PODATAKA .....   | 3         |
| 2.3      | PROGRAMIRANJE KONZOLNE APLIKACIJE U SVRHU TESTIRANJA PERFORMANSI SKLADIŠTA   | 3         |
| <b>3</b> | <b>OPIS KORIŠĆENIH TEHNOLOGIJA I ALATA .....</b>   | <b>4</b>  |
| 3.1      | TEHNOLOGIJE I ALATI KORIŠĆENI ZA POTREBE DEFINISANJA MODELA DISTRIBUTIVNOG IZVODA.....   | 4         |
| 3.2      | TEHNOLOGIJE I ALATI KORIŠĆENI ZA POTREBE KREIRANJA APLIKACIJE SA SPREGOM SA RDF SKLADIŠTEM PODATAKA I APLIKACIJE ZA TESTIRANJE PERFROMANSI SKLADIŠTA ..... | 5         |
| <b>4</b> | <b>OPIS REŠENJA PROBLEMA .....</b>   | <b>6</b>  |
| 4.1      | DEFINISANJE MODELA DISTRIBUTIVNOG IZVODA NA OSNOVU CIM PROFILA .....   | 6         |
| 4.1.1    | UČITAVANJE CIM MODELA U ENTERPRISE ARCHITECT I GENERISANJE XMI-ja.....   | 6         |
| 4.1.2    | UČITAVANJE XMI FAJLA U ECLIPSOV CIMTOOL ALAT I GENERISANJE CIM PROFILA .....   | 7         |
| 4.1.3    | GENERISANJE INSTANCI PODATAKA KORISTEĆI CIMET ALAT .....   | 12        |
| 4.2      | C# .NET APLIKACIJA ZA UČITAVANJE I SKLADIŠTENJE CIM MODELA I PRIMERA PODATAKA  | 13        |
| 4.2.1    | KORISNIČKI INTERFEJS .....   | 13        |
| 4.2.2    | UČITAVANJE I PARSIRANJE CIM PROFILA I CIM PRIMERA PODATAKA.....  | 14        |
| 4.2.3    | FUSEKI RDF SKLADIŠTE I NJEGOVA SPREGA SA APLIKACIJOM UZ POMOĆ dotNETRDF BIBLIOTEKE.....  | 16        |
| 4.3      | C# .NET APLIKACIJA ZA POTREBE TESTIRANJA PERFORMANSI JENA RDF SKLADIŠTA I RELACIONIH BAZA.....   | 22        |
| 4.3.1    | TESTIRANJE PERFORMANSI JENA RDF SKLADIŠTA .....  | 22        |
| 4.3.2    | TESTIRANJE PERFORMANSI RELACIONIH BAZA PODATAKA.....   | 26        |
| 4.3.3    | REZULTATI MERENJA PERFORMANSI JENA RDF, mySQL i PostgreSQL SKLADIŠTA I NJIHOVO UPOREĐIVANJE .....  | 32        |
| <b>5</b> | <b>PREDLOZI ZA DALJA USAVRŠAVANJA .....</b>  | <b>38</b> |
| <b>6</b> | <b>LITERATURA.....</b>   | <b>40</b> |

## 2 OPIS REŠAVANOG PROBLEMA

Osnovni zadatak projekta predstavlja pronalaženje načina standardizovanog modelovanja jednog primera distributivnog izvoda, odnosno dela električne mreže koji se nalazi u okviru podsistema distribucije, a potom i kreiranje softverske aplikacije koja će omogućiti učitavanje, skladištenje i postavljanja upita nad takvim reprezentativnim modelom i na kraju testiranje performansi samog RDF skladišta tzv. Triplestore-a i njegovo poređenje sa performansama relacionih baza podataka u slučajima gde se radi sa RDF fajlovima različitih veličina.

Primer distributivnog izvoda koji se zahteva u okviru ovog projekta je IEEE 37 NODE Test Feeder, koji može da se nađe na [1]. Test Feeder može da se zamisli kao model distributivne mreže, koji je u stanju da replicira ponašanje realnog distributivnog fidera [2], a koji je stvoren sa svrhom simuliranja različitih uslova i stanja kako bi se povećala pouzdanost, efikasnost i performanse distributivne mreže ili jednog njenog dela.

Primeri RDF fajlova koji će se koristiti za proveru performansi RDF skladišta i relacionih baza podataka se nalaze na [3].

Uzimajući prethodno izlaganje u obzir, celokupno rešavanje problematike zadatka može se podeliti na 3 celine

- Definisanje modela distributivnog izvoda
- Programiranje aplikacije koja će omogućiti učitavanje modela i primera (instance) podataka, a zatim i spregu sa skladištem koje je pogodno za reprezentaciju komponenti distributivne mreže kao i odnosa između tih komponenti
- Programiranje konzolne aplikacije koja će omogućiti testiranje performansi skladišta

### 2.1 DEFINISANJE MODELA DISTRIBUTIVNOG IZVODA

Ovo podrazumeva definisanje informacionog modela koji je reprezentacija koncepata, njihovih odnosa, ograničenja, pravila i operacija koji specificiraju semantiku podataka u okviru domena elektroenergetskih sistema [3].

Daleko najzastupljeniji informacioni model koji se koristi kod postojećih rešenja, pa tako i ovog rešenja je Common Information Model odnosno CIM.

CIM je otvoren standard za reprezentaciju komponentata energetskog sistema koji je prvobitno razvijen od strane Instituta za istraživanje elektro energetike (eng. EPRI), sada pod pokroviteljstvom Međunarodne elektrotehničke komisije (IEC). Radna grupa 13 tehničkog komiteta IEC 57 održava osnovni CIM model kao jezički nezavisan UML model, definišući komponente energetskog sistema kao klase zajedno sa odnosima između ovih klasa: nasleđivanje, asocijaciju i agregaciju; takođe su definisani i parametri unutar svake klase [4].

U slučaju ovog projekta da bi definisali model našeg distributivnog izvoda, na osnovu pomenutog UML modela neophodno je definisati podskup modela koji se odnosi na domen IEEE 37 NODE Test Feeder-a, odnosno potrebno je definisati CIM Profil. Profili se mogu predstaviti u različitim oblicima, uključujući RDFS, XML Šemu, tekstualne dokumente ili HTML. [5].

Kao finalni korak dobijeni CIM Profil koji je predstavljen nekim od gore pomenutih oblika potrebno je iskoristiti kako bi napravio fajl primera instanci podataka (CIM/XML ili RDF/XML) i popuniti ga podacima koji odgovaraju informacijama dostupnim u okviru dokumentacije test fidera.

Iako se CIM smatra zlatnim standardom u ovoj oblasti, postoje i alternativni pristupi od kojih treba pomenuti MultiSpeak.

Pristup koji je usvojen od strane MultiSpeak-a bio je da definiše interfejs umesto da definiše zajednički, sveobuhvatni model podataka sa kojim bi svi dobavljači morali da se usaglase. Ovaj pristup omogućava dobavljačima da pišu i održavaju interfejs koji prikuplja potrebne informacije iz njihove osnovne strukture podataka, konvertuje te podatke u XML (Extensible Markup Language) podatke i prenosi te pakete u obliku unapred definisanih poruka. Aplikacija koja prima podatke je odgovorna za razdvajanje poruke, parsiranje XML-a i preduzimanje odgovarajuće radnje sa rezultirajućim objektima podataka [6]. Na ovaj način može se videti da je cilj MultiSpeak-a takođe standardizovan način razmene podataka između komponenti distributivnog podsistema.

## 2.2 PROGRAMIRANJE APLIKACIJE I SPREGA SA SKLADIŠTEM PODATAKA

U ovom koraku potrebno je programirati aplikaciju koristeći neki od RDF frejmworka koji su pogodni za rad sa RDF tipom podataka. Kao što je već navedeno, RDF/RDFS/XML oblici fajlova imaju informacije o našoj ontologiji (CIM Profil) ili primeru instanciranih podataka test fidera.

Uz pomoć API-ja ovih biblioteka neophodno je zatim obezbediti mogućnost učitavanja, parsiranja i skladištenja RDF trojki, a potom i komunikaciju sa nekim od specijalizovanih RDF skladišta, tako što će omogućiti operacije nad strukturama podataka koje se koriste za rad so trojkama, a to su najčešće grafovi. Operacije koje se implementiraju su:

- Sačuvati graf u skladištu
- Obrisati graf iz skladišta
- Modifikovati (dodati ili obrisati) neku od trojki grafa i tu izmenu sačuvati u okviru skladišta
- Učitati graf iz skladišta radi njegovog prikaza u okviru aplikacije

I kao poslednji korak potrebno je omogućiti korisniku mogućnost postavljanja upita u odnosu na učitani model i podatke referencirajući RDF skladište, kako bi mogao dobiti određene informacije o stanju distributivne mreže.

Trenutno je prisutan veliki broj biblioteka koje omogućavaju rad sa RDF-om i te biblioteke se razlikuju u odnosu na to za koji programski jezik su namenjene, jedan primer za Java programski jezik je Apache Jena.

Ova biblioteka sadrži RDF API za kreiranje, čitanje i rad sa RDF grafovima na nivou trojki, kao i mogućnost serijalizacije trojki uz pomoć nekog popularnog formata kao što je RDF/XML ili turtle.

Takođe sadrži API za generisanje upita koristeći SPARQL query jezik putem ARQ engine-a, a potom i API za rad sa skladištem podataka putem Fuseki ili tdb (tdb2).

Fuseki omogućava skladištenje i postavljanje upita nad RDF podacima preko WEB-a, dok tdb omogućava skladištenje na disku s tim da je daleko skalabilnije rešenje i omogućava čuvanje velikog broja grafova, izuzetno je visokih performansi, a takođe podaci sačuvani u ovom tipu skladišta su trajni.

Postoji i Ontology API koji je fokusiran na rad sa RDFS ili OWL modelima sa ciljem dodavanja određenih semantičkih značenja RDF podacima, dok Inference API omogućava donošenje dodatnih zaključaka na osnovu priloženog modela i podataka koji nisu eksplicitno definisani. Više informacija o Apache Jena se nalaze na [7].

Neki od ostalih primera su RDFLib (Python), Stardog, Sesame (Java) , Virtuoso itd.

Ovaj projekat se bazira na korišćenju .NET C# biblioteke koja ima API za rad sa RDF podacima dotNetRDF i korišćenjem njenog API-ja za spregu sa Apache Jena Fuseki SPARQL Web Serverom kao skladištem podataka.

## 2.3 PROGRAMIRANJE KONZOLNE APLIKACIJE U SVRHU TESTIRANJA PERFORMANSI SKLADIŠTA

Ovaj korak podrazumeva kreiranje konzolne aplikacije u okviru koje će se izvršavati testiranje performansi skladišta specijalizovanih za rad sa RDF trojkama (eng. Triplestore), a zatim i njihovo poređenje sa performansama standardnih relacionih baza podataka pri radu sa RDF fajlovima koji sadrže različit broj RDF trojki.

Prilikom testiranja performansi potrebno je prikazati sledeće informacije:

- Specifikacija računara na kojem je testiranje vršeno
- Verzija korišćenog frejmworka
- Srednje vreme brzine izvršavanja operacija
- Opseg greške odstupanja od svih merenja
- Memorijsko zauzeće

Testiranje je fokusirano na prethodno implementirane CRUD (Create, Read, Update, Delete) operacije, kao i testiranje performansi izvršavanja postavljenih upita nad skladištem.

U slučaju Jena RDF skladišta, koje je jedno od komercijalno dostupnih Triplestore skladišta, postoje dva načina inicijalizacije dataseta korišćenjem Apache Jena Fuseki-ja koje je neophodno testirati:

- In-memory dataset
- tdb2 dataset

Dataset predstavlja kontejner u okviru koga se mogu smestiti RDF fajlovi.

In-memory dataset predstavlja kontejner gde se dolazeći RDF fajlovi smeštaju u RAM memoriju računara i tako smešteni fajlovi nisu perzistentni, odnosno drugim rečima biće tu prisutni dok god je Fuseki server pokrenut i nestaju prilikom njegovog gašenja, dok tdb2 dataset predstavlja kontejner gde se RDF fajlovi smeštaju na disku, u okviru tdb2 baze i perzistentni su i u slučaju gašenja Fuseki servera.

U slučaju relacionih baza podataka neophodno je ispitati sve prethodno navedene operacije i napraviti ekvivalentan model na osnovu koga će se kreirati tabele u okviru kojih će biti smešteni RDF grafovi i RDF trojke, obezbediti konekciju aplikacije sa bazom podataka, a zatim i izvršiti tražena merenja performansi.

Merenje performansi za slučaj relacionih baza podataka potrebno je proveriti u okviru mySQL i PostgreSQL skladišta.

### 3 OPIS KORIŠĆENIH TEHNOLOGIJA I ALATA

#### 3.1 TEHNOLOGIJE I ALATI KORIŠĆENI ZA POTREBE DEFINISANJA MODELA DISTRIBUTIVNOG IZVODA

- **Enterprise Architect** – veoma jednostavan alat koji pre svega podržava UML modelovanje koje je usko povezano sa CIM-om, pored toga jedna od glavnih prednosti je visok stepen mogućnosti prilagođavanja modela ličnim potrebama (npr. korišćenje stereotipa koji bliže opisuju koncept ili neko svojstvo), kao i mogućnost generisanja standarda za razmenu XML meta podataka (XML) na osnovu kojeg se u ostalim alatima može praviti podskup modela
- **Eclipse** – alat koji predstavlja razvojno okruženje koje je između ostalog i okruženje za rad sa raznim modelima kao što je UML model, odnosno sadrži plugin CIM Tool koji prihvata i radi sa šemom UML modela
- **CIM Tool** – zvaničan alat neophodan za dizajn CIM profila na bazi UML šeme modela date u XML formatu, korišćen zbog definisanja modela u domenu IEEE 37 NODE Test Feeder-a. Veoma pogodan da na osnovu prethodno pomenute šeme pruži veoma intuitivan postupak izabira klasa, atributa, definicija restrikcija itd. koji će sačinjavati naš CIM Profil, a zatim i generisanje izlaznih RDFS, OWL, HTML fajlova koji sadrže definiciju modela nama od interesa
- **CIMET** – alat koji omogućava da se na osnovu definisanog CIM Profila generiše šablonski RDF ili XML fajl instanca naših klasa, atributa itd. koji se zatim popunjava realnim podacima koji odgovaraju traženom test fideru
- **UML** – jezik korišćen u svrhu modelovanja raznoraznih sistema, pogodan za ovaj projekat prvenstveno jer je celokupan CIM data model održavan kao UML model, izuzetno jasno omogućava definisanje, vizuelizaciju, dokumentaciju CIM komponenti, njihovih atributa, odnosa između komponenti itd. Pomaže da se bliže opiše struktura i ponašanje našeg test fidera distributivnog sistema koristeći UML dijagrame od kojih je najčešće korišćen dijagram klasa
- **RDF/RDFS/XML** – RDF i RDFS predstavljaju semantičke web tehnologije izabrane jer odgovaraju bližem opisivanju značenja komponenti, atributa itd. CIM modela, uz pomoću njihovog vokabulara, takođe ovi dokumenti su kompatibilni za razmenu podataka između različitih aplikacija, različitih tipova računara odnosno predstavljanje CIM data modela putem njih je standardizovano, poseduju i vokabular za opis ontologija, sposobnost izvođenja dodatnih zaključaka o podacima, lako mogu biti prošireni dodatnim CIM elementima itd. XML se koristi kao tehnologija serijalizacije RDF i RDFS pre svega zbog svoje čitljive i lako razumljive strukture kao i lakoće parsiranja XML dokumenata



### 3.2 TEHNOLOGIJE I ALATI KORIŠĆENI ZA POTREBE KREIRANJA APLIKACIJE SA SPREGOM SA RDF SKLADIŠTEM PODATAKA I APLIKACIJE ZA TESTIRANJE PERFROMANSI SKLADIŠTA

- **Visual Studio 2019** – predstavlja veoma bogato integrisano razvojno okruženje, pored toga što podržava širok spektar frejmworkova i jezika, ima i odlične preporuke pisanja koda uz pomoć svog IntelliSensa, projektna šablona, detaljno debugovanje (watch windows, breakpoints, call stacks itd.), integrisanje s ostalim alatima kao što je GIT
- **C# .NET** – pre svega izabran zbog upoznatosti sa funkcionisanjem frejmworka i velikog izbora različitih korisnih biblioteka od kojih je jedna i dotNetRDF, prednosti su još i automatsko rukovanje memorijom (garbage collector), dobrih performansi prilikom izvršavanja koda (runtime) korišćenjem just-in-time kompajliranja
- **dotNetRDF biblioteka** – izabrana kao biblioteka koja radi u .NET okruženju i pruža mogućnost rukovanja sa RDF podacima, od učitavanja u graf strukturu do njegovog parsiranja i sprege sa već implementiranim komercijalnim RDF skladištima podataka, sadrži i API-je koje omogućavaju formatiranje podataka, izradu konfiguracionih fajlova, rad sa SPARQL upitima itd.
- **Windows Forms** – je tehnologija koja omogućava kreiranje grafičkog korisničkog interfejsa u okviru već izabranog .NET frejmworka, prednost je u tome što omogućava brzo i jednostavno, koristeći svoj toolbox drag & dropom metodom kreiranje tipičnih kontrola GUI-ja kao što su dugmad, tekst polja, a potom na osnovu event-ova logiku rukovanja, podržava i data binding
- **SPARQL** – je jezik za postavljanje upita nad RDF strukturom podataka, pogodno je to što je veoma analogan SQL jeziku koji se koristi u relacionim bazama podataka
- **Apache Jena Fuseki Server/Storage** – koristi se kao bekind skladište podataka u slučaju kada imamo RDF tipove podataka, pa se samim tim nameće se kao jedno od logičkih rešenja, pored toga što služi kao skladište fuseki je i SPARQL web server i omogućava slanje SPARQL upita putem HTTP protokola tako što izlaže tačke pristupa serveru koji se nazivaju servisi
- **BenchmarkDotNet biblioteka** - je biblioteka koja omogućuje testiranja različitih delova koda ili funkcija u okviru .NET okruženja, prednost biblioteke je to što uzima u obzir Just-In-Time(JIT) kompajliranje i garbage collection priliom merenja performansi čime se dobijaju preciznije vrednosti, takođe frejmwork BenchmarkDotNet biblioteke omogućava da se vrši višestruko testiranje željenog dela koda čime se obezbeđuje da su merenja konstantna i da nisu rezultat nasumične varijacije, sadrži i bogat i informativan spektar izveštaja rezultata koristeći statistiku, grafikone, kao i druge načine vizuelizacije koja pomažu u tumačenju rezultata
- **Entity Framework** – je frejmwork koji služi za objektno-relaciono mapiranje za .NET aplikacije, drugim rečima omogućava mapiranje između objektno-orijentisanog programskog jezika i relacionih baza podataka i rad sa entitetima baze podataka koristeći C# .NET objekte, takođe pogodno je i to što omogućava rad sa mnoštvom komercijalnih relacionih baza podataka u isto vreme bez mnogo velikog uticaja na menjanje aplikacionog koda, zatim omogućava automatsko generisanje šeme baze podataka na osnovu modela entiteta C# aplikacije i pruža LINQ jezik pomoću kojeg se programski mogu postavljati upiti u C# jeziku
- **MySQL Workbench** – predstavlja alat za dizajniranje i upravljanje MySQL bazom podataka sa grafičkim korisničkim interfejsom, odnosno omogućava vizuelni rad sa MySQL bazom podataka, pored ove osnovne funkcije ovaj alat pruža mogućnost kreiranja i izvršavanja SQL upita i skripti uz pomoć ugrađenog SQL editora koji pruža funkcije naglašavanja sintakse, formatiranje upita itd. sadrži i administrativne alate za upravljanje, konfigurisanje i praćenje performansi instanci MySQL servera
- **PostgreSQL pgAdmin4** – predstavlja alat za upravljanje PostgreSQL bazom podataka sa grafičkim korisničkim intefejsom, odnosno omogućava vizuelni rad sa PostgreSQL bazom podataka, kao i u slučaju MySQL Workbencha sadrži SQL editor za kreiranje i izvršavanje SQL komandi nad PostgreSQL bazom, administrativne mogućnosti kao i sve operacije manipulacija nad bazom podataka kao što su kreiranje, modifikovanje i brisanje baze, tabela, indeksa korišćenjem grafičkog interfejsa

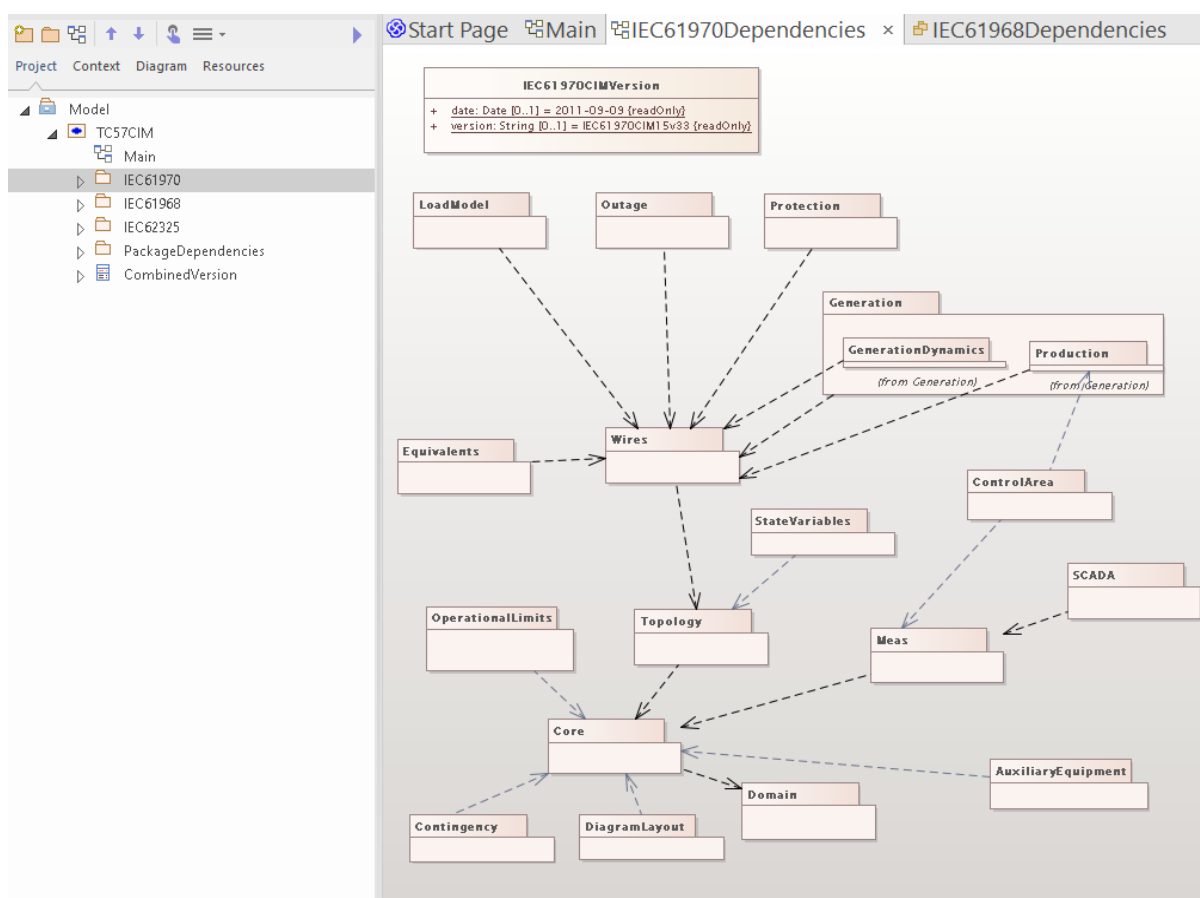
## 4 OPIS REŠENJA PROBLEMA

### 4.1 DEFINISANJE MODELA DISTRIBUTIVNOG IZVODA NA OSNOVU CIM PROFILA

#### 4.1.1 UČITAVANJE CIM MODELA U ENTERPRISE ARCHITECT I GENERISANJE XMI-ja

Početni korak u definisanju modela distributivnog izvoda odnosi se na učitavanje CIM UML modela u okviru Enterprise Architect alata. Ovaj model, koji je dostupan organizacijama kao “gotov” proizvod za preuzimanje, sastoji se od paketa IEC 61970, IEC 61968, IEC 62325.

IEC standard 61970-301 je semantički model koji opisuje komponente elektroenergetskog sistema na električnom nivou i odnose između svake komponente. IEC 61968-11 proširuje ovaj model kako bi obuhvatio i druge aspekte razmene podataka elektroenergetskog sistema, kao što su praćenje rada komponenti, planiranje poslova i naplate usluga korisniku [8]. CIM za elektroenergetsko tržište zatim proširuje oba ova modela sa IEC 62325-3013 kako bi obuhvatio podatke razmenjene između učesnika na elektroenergetskom tržištu [9].



Slika 3.1 Primer izgleda CIM IEC 61970 UML modela u Enterprise Architect alatu

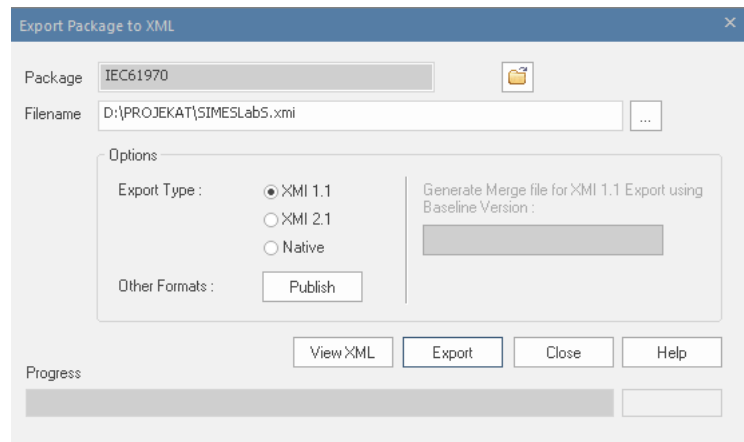
Na slici 3.1 je jedan primer izgleda CIM UML modela, u ovom slučaju za paket IEC 61970. Sastoji se od podskupa paketa, prikazanim na slici pravougaonim oblikom, koji modeluju jedan deo ovog IEC 61970 paketa. Strelicama su prikazane veze zavisnosti između paketa.

Na primer paket Core modeluje osnovne klase, attribute i odnose između klasa, kao što je klasa `PowerSystemResource` koja reprezentuje bilo koji entitet elektroenergetskog sistema, nju nasleđuje nešto specifičnija klasa `Equipment` koja modeluje bilo koji fizički, električni ili mehanički uređaj elektroenergetskog sistema. Veza zavisnosti, na primer između paketa Core i paketa Domain označava da paket Core zavisi od paketa Domain. U ovom konkretnom smislu paket Domain sadrži sve neophodne osnovne tipove podataka koje paket Core koristi da bi ih pridružio određenim entitetima i tako bliže opisao (modelovao) taj entitet.

Nakon što je CIM model učitao u EA alat, da bi se dalje mogao razmenjivati između alata ili softverskih sistema on se transformiše u XML format. XML sam po sebi je standard za razmenu meta podataka

odnosno podataka o podacima, on opisuje karakteristike i daje kontekst podacima informacionog modela (CIM). XML dokument takođe se može pregledati u tekstualnom editoru, a sa druge strane biti pročitani nekim od alata za modeliranje.

Enterprise Architect nudi mogućnost generisanja XML fajla na osnovu UML modela, što je u ovom slučaju CIM. Potrebno je u okviru Project prozora (pogledati sliku 3.1) selektovati korenski element, a zatim iz glavnog menija izabrati opciju **Publish** ➡ **Export XML** ➡ **Export XML for current package**.



Slika 3.2 Izgled dijaloga prilikom biranje opcije generisanja XML-ja

Nakon biranja ove opcije prikazaće se dijalog koji se može videti na slici 3.2. Parametar **Package** označava selektovani korenski paket u okviru kojeg se nalaze sve definisane klase, atributi i veze (asocijacije, agregacije, kompozicije) koje će biti obuhvaćene u XML fajlu.

**Export Type** parametrom se bira verzija XML dokumenta koje će biti korišćena u okviru njegovog generisanja, dodatne verzije/formati mogu se pregledati biranje opcije **Publish**.

**View XML** opcija omogućava pregled XML dokumenta pre njegovom neposrednog generisanja.

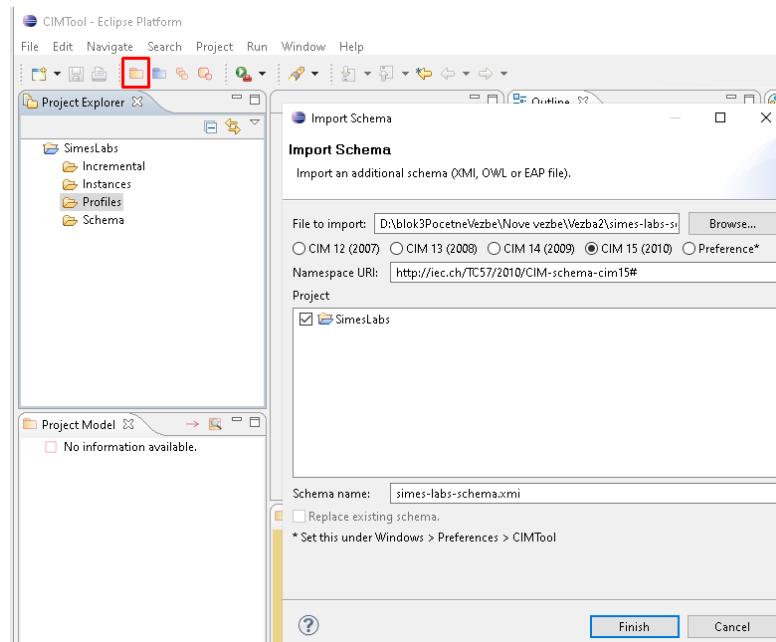
**Export** opcijom generišemo XML dokument, fajl će smešten na putanji koja je navedena u okviru **Filename** parametra.

#### 4.1.2 UČITAVANJE XMI FAJLA U ECLIPSOV CIMTOOL ALAT I GENERISANJE CIM PROFILA

Sledeći deo odnosi se na učitavanje XML fajla u program Eclipse, odnosno korišćenje njegovog posebnog plugin-a CIM Tool. CIM Tool nam omogućava definisanje CIM profila tj. kontekstno zavisnog modela. Drugim rečima svi delovi CIM modela su opcioni u zavisnosti od aplikacije i njenih zahteva, samim tim iz ovog proizlazi neophodnost dokumenta koji specificira samo neophodne delove, a to je CIM Profil.

Kako bi kreirali CIM profil potrebno je iz glavnog menija izabrati opciju **File** ➡ **New** ➡ **CIMTool Project** i dati naziv projekta.

Nakon ovoga u Project Explorer prozoru pojaviće se kreirani projekat. Dalje potrebno je učitati XML šemu u projekat, pogledati sliku 3.3.



Slika 3.3 Izgled dijaloga nakon biranja opcije učitavanja XML šeme

Opcija koja omogućava učitavanje XML šeme uokvirena je na slici 3.3 crvenim kvadratom, nakon biranja ove opcije pojaviće se dijalog za učitavanje šeme.

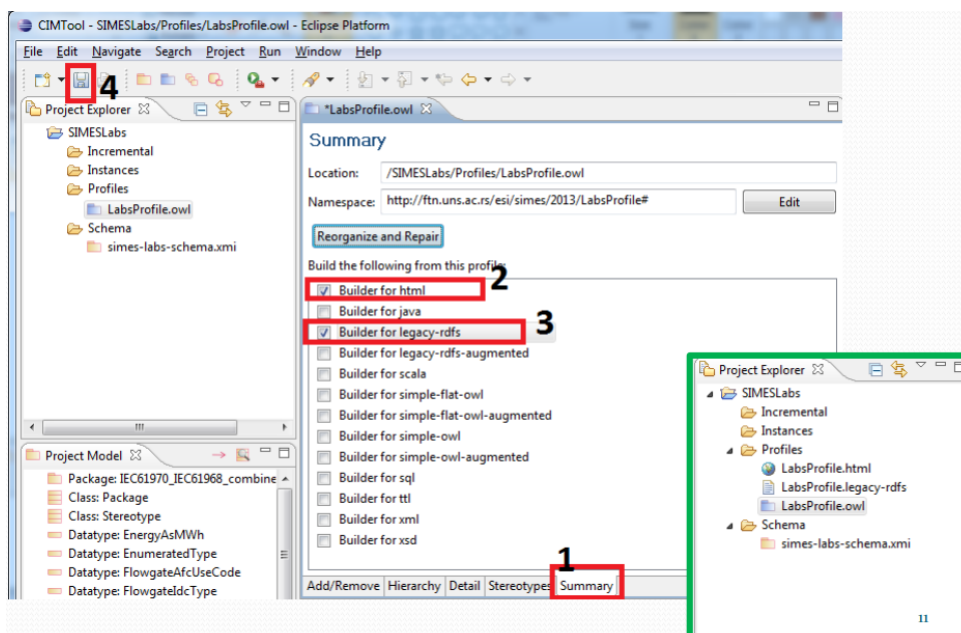
**File to import** parametar predstavlja putanju do XML fajla. Biranjem fajla automatski će biti ponuđeno ime šeme u okviru parametra **Schema name** koje se može modifikovati po želji.

Takođe potrebno je izabrati jednu od opcija radio dugmeta, gde svaka od njih označava verziju CIM modela, CIM 15 označava CIM Model iz 2010, biranjem jedne od opcija automatski će se ažurirati i **Namespace URI** parametar. Opcija **Preference\*** predstavlja generičku verziju.

Nakon popunjavanja svih ovih parametara konačno klikom na **Finish** XML model će biti učitani u naš CIMTool projekat.

Nakon što smo učitali XML šemu u okviru projekta neophodno je započeti sa definisanjem CIM profila. Potrebno je u okviru glavnog menija izabrati opciju **File** ➡ **New** ➡ **CIMTool Profile** i dodeliti ime profilu i kliknuti na **Finish**.

U okviru Profiles foldera pojaviće se kreirani CIM profil u .owl formatu. Moguće je izabrati i neki drugi format za zapis CIM profil prateći korake sa slike 3.4, koja konkretno pokazuje dodavanje html i .legacy-rdfs formata.



Slika 3.4 Izgled projekat nakon dodavanja profila i biranja formata zapisa CIM profila

U sledećem koraku sledi unos neophodnih klasa, atributa, definisanje veza (referenci, kardinaliteta). Izbor navedenog u direktnoj je zavisnosti od distributivnog izvoda odnosno u ovom slučaju IEEE Node 37 Test Feeder-a.

Na osnovu takozvanog One line dijagrama potrebno je modelovati test fider u kontekstu CIM-a. One line dijagram je prikazan na slici 3.5.

U okviru ovog dijagrama i na osnovu pravila modelovanja koja se nalaze na [10], treba uočiti da deo označen punom linijom označava provodnik uz pomoć kojeg se prenosi naizmenična struja, a to se referencirajući se na pravila modelovanja u CIM-u modeluje klasom **ACLineSegment**.

Brojevima su na dijagramu označeni potrošači koji se povezani na distributivnu mrežu i oni se modeluju sa CIM klasom **EnergyConsumer**.

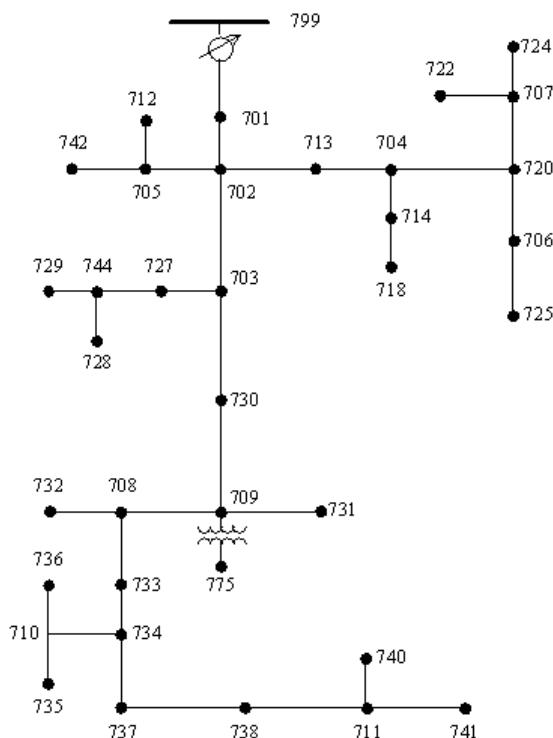
CIM takođe definiše konekciju između različitih delova sistema. Umesto direktne konekcije, kao na primer konekcije između provodnika i potrošača 1 na 1, CIM koristi pomoćne klase **ConnectivityNode** i **Terminal**. Razlog ovakvog pristupa jer bi tada u slučaju više komponenti za svaku od njih morala bi da postoji referenca na preostale komponente kako bi se modelovala njihova povezanost.

Korišćenjem ConnectivityNode sve ove komponente imaju referencu ka tom ConnectivityNode umesto toga. Treba napomenuti da komponente nisu direktno povezane sa ovim čvorovima već koriste pristupne tačke koje se nazivaju Terminali. Osim toga što služe za modelovanje pristupnih tačaka, terminali služe kao i tačke gde su moguća merenja struje i napona.

Treba primetiti i prisustvo transformera. On se u okviru CIM-a modeluje klasom **PowerTransformer**, dok se namotaji modeluju klasom **PowerTransformerEnd**.

Zatim treba uzeti u obzir i apstraktne klase kao što su **IdentifiedObject** koja se koristi za identifikovanje komponente tako što se komponenti dodeljuje jedinstveni ID i ime izabrano tako da je razumljivo za čoveka koji taj dokument može pročitati. **Equipment** i **ConductingEquipment**, koje modeluju električnu opremu, odnosno provodnu opremu itd.

Ovim klasama se dodeljuju određeni atributi, ponovo u zavisnosti od distributivnog izvoda, neki primeri atributa mogu biti **r** za klasu **ACLineSegment** koja definiše otpor protoku naizmenične struje kroz taj provodnik. **CustomerCount** za **EnergyConsumer** klasu koji definiše broj korisnika povezan na potrošač kojem se doprema usluga snabdevanja električnom energijom itd.



Slika 3.5 One line dijagram IEEE 37 Node Test Feeder-a

Nakon identifikacije svih neophodnih klasa, atributa i odnosa preostaje još da se oni dodaju u okviru CIM Tool alata. Slede primeri dodavanja jedne klase, jednog atributa i jedne reference, ovo je

neophodno ponoviti i za sve preostale klase, attribute i reference.

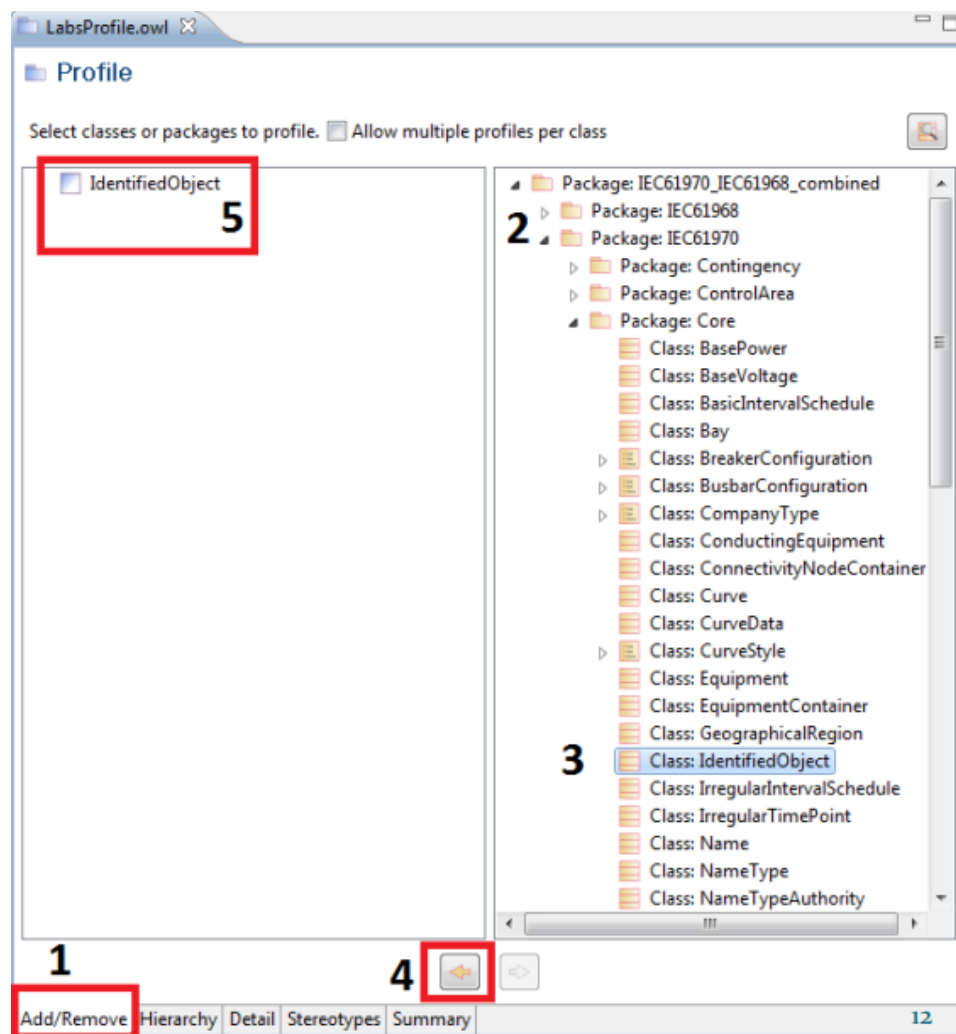
Na slici 3.6 je prikazan primer dodavanja jedne klase, u ovom slučaju klase IdentifiedObject.

U koraku 1 neophodno je izabrati opciju Add/Remove koja omogućava dodavanje odnosno uklanjanje klase iz CIM Profila

U koraku 2 potrebno je izabrati željenu klasu iz paketa, u slučaju klase IdentifiedObject to je paket Core u okviru Paketa IEC61970.

U koraku 3 i 4 selektujemo željenu klasu i klikom na označenu strelicu dajemo komandu da ta klasa postaje deo CIM profila.

U koraku 5 se vidi prikaz svih trenutno dodatih klasa.



Slika 3.6 primer dodavanja jedne klase u CIM profil

Na slici 3.7 je prikazan primer dodavanja jednog atributa, u ovom slučaju atributa mRID.

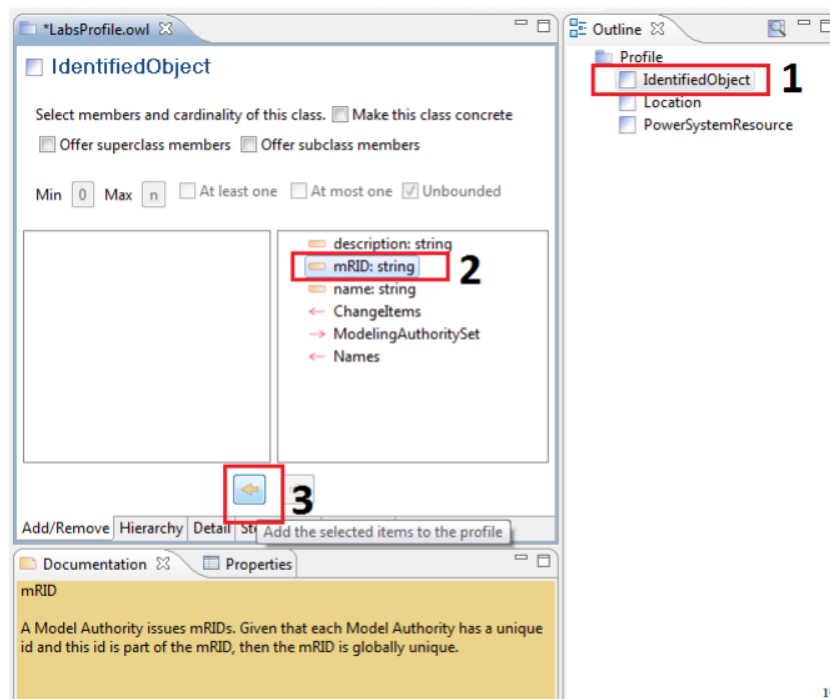
U koraku 1 potrebno je izabrati klasu u okviru koje se bira atribut koji će biti uključen u CIM profil.

U koraku 2 selektuje se željeni atribut i u koraku 3 klikom na označenu strelicu atribut postaje deo CIM profila.

Još neke od opcija koje su moguće su konfiguracija klase tj. biranje da li je klasa apstraktna ili konkretna, tako što se otkači opcija Make this class concrete, ako označimo da je klasa konkretna dalje je moguće ograničiti moguć broj instanci ove klase. Apstraktna klasa se ne može biti instancirana.

Atributi koji su enumeracije neophodno je pored dodavanja kao atributa, dodati i kao klasu.

Takođe u tabu Documentation dat je kratak opis značenja atributa.



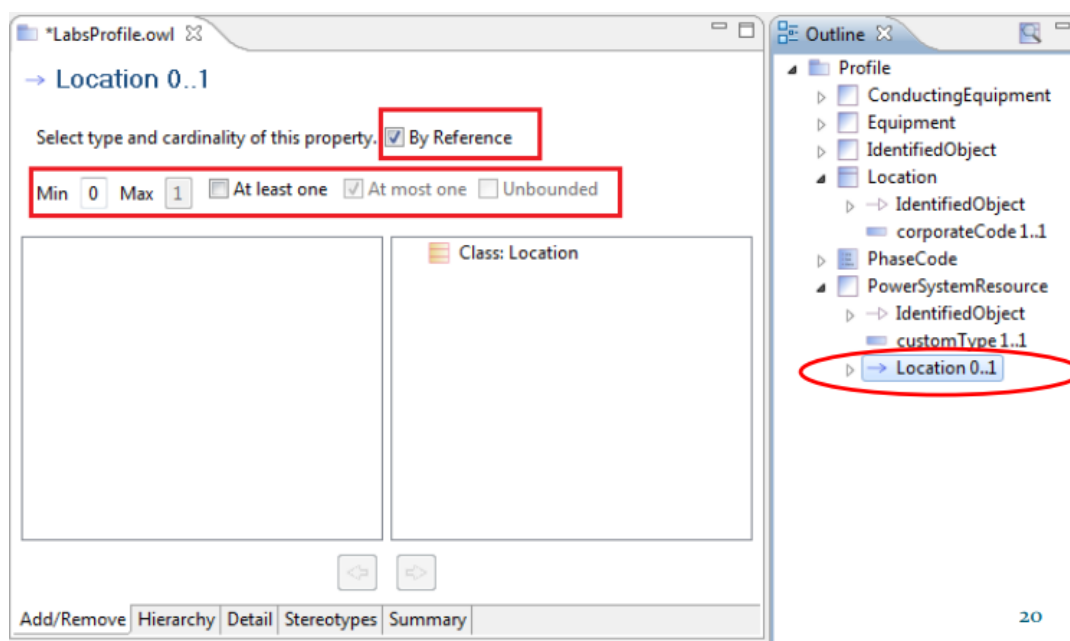
Slika 3.7 primer dodavanja jednog atributa u CIM profil

Na slici 3.8 dat je primer dodavanja jedne reference, u ovom slučaju Location u okviru PowerSystemResource klase.

Referencu prvobitno dodajemo kao i svaki atribut kao što je prikazano na slici 3.7, sa dodatkom da je nakon toga neophodno selektovati taj isti atribut i otkačiti polje By Reference čime definišemo da je taj atribut tipa reference.

Preostaje još samo definisati kardinalitet, odnosno broj pojavljivanja određenog atributa u okviru neke klase.

Nakon svega ovoga moguće je pregledati CIM profil fajl u nekome od formata zapisivanja koje su izabrani.



Slika 3.8 primer dodavanja jedne reference u CIM profil

## Kratik pregled jednog dela CIM profila dat je na slici 3.9

```
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
  xmlns:cims="http://iec.ch/TC57/1999/rdf-schema-extensions-19990926#"
  xmlns:uml="http://langdale.com.au/2005/UML#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
  xml:base="http://iec.ch/TC57/2010/CIM-schema-cim17#"
  <rdf:Description rdf:about="#ACLineSegment">
    <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-schema#Class"/>
    <rdfs:label>ACLineSegment</rdfs:label>
    <rdfs:comment>A wire or combination of wires, with consistent electrical characteristics, building
    For symmetrical, transposed three phase lines, it is sufficient to use attributes of the line segment, which descri
    The BaseVoltage
    at the two ends of ACLineSegments in a line shall have the same BaseVoltage.nominalVoltage. However
    <cims:belongsToCategory rdf:resource="#Package_Wires"/>
    <rdfs:subClassOf rdf:resource="#ConductingEquipment"/>
  </rdf:Description>
  <rdf:Description rdf:about="#ACLineSegment.PerLengthImpedance">
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
    <rdfs:label>PerLengthImpedance</rdfs:label>
    <rdfs:comment>Per-length impedance of this line segment.</rdfs:comment>
    <rdfs:domain rdf:resource="#ACLineSegment"/>
    <rdfs:range rdf:resource="#PerLengthImpedance"/>
    <cims:multiplicity rdf:resource="http://iec.ch/TC57/1999/rdf-schema-extensions-19990926#M:0..1"/>
    <cims:inverseRoleName rdf:resource="#PerLengthImpedance.ACLineSegments"/>
  </rdf:Description>
  <rdf:Description rdf:about="#ACLineSegment.WireSpacingInfo">
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
    <rdfs:label>WireSpacingInfo</rdfs:label>
    <rdfs:domain rdf:resource="#ACLineSegment"/>
    <rdfs:range rdf:resource="#WireSpacingInfo"/>
    <cims:multiplicity rdf:resource="http://iec.ch/TC57/1999/rdf-schema-extensions-19990926#M:0..1"/>
    <cims:inverseRoleName rdf:resource="#WireSpacingInfo.ACLineSegment"/>
  </rdf:Description>
  <rdf:Description rdf:about="#ACLineSegment.b0ch">
    <rdf:type rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#Property"/>
    <rdfs:label>b0ch</rdfs:label>
    <rdfs:comment>Zero sequence shunt (charging) susceptance, uniformly distributed, of the entire line
    <rdfs:domain rdf:resource="#ACLineSegment"/>
    <cims:dataType rdf:resource="#Susceptance"/>
    <cims:multiplicity rdf:resource="http://iec.ch/TC57/1999/rdf-schema-extensions-19990926#M:0..1"/>
    <cims:stereotype rdf:resource="http://langdale.com.au/2005/UML#attribute"/>
  </rdf:Description>
```

Slika 3.9 primer izgleda dela CIM profila u .legacy-rdfs formatu

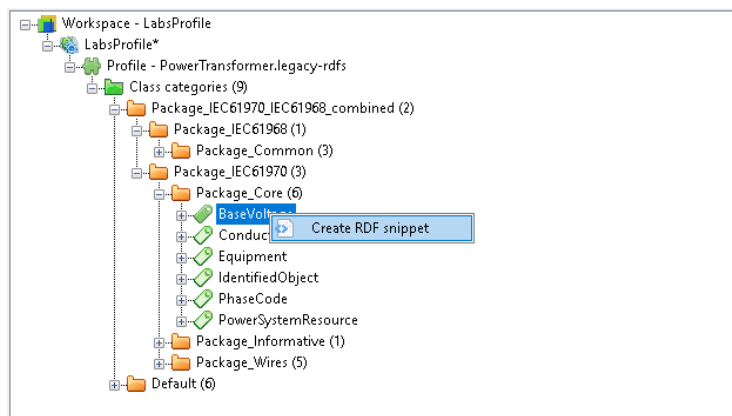
U prvom delu dokumenta definisani su imenski prostori koji predstavljaju opseg definisanosti identifikatora koji su povezani sa tim imenskim prostorom.

Sledi definisanje klasa čije telo sadrži njihove pridružene attribute i reference. RDFS dokument uz pomoć svog vokabulara daje značenje ovim klasama i atributima odnosno pruža opis meta podataka.

### 4.1.3 GENERISANJE INSTANCI PODATAKA KORISTEĆI CIMET ALAT

Predstavlja finalni korak u definisanju distributivnog izvoda. On omogućava da se na osnovu učitano CIM Profila izgeneriše šablon za kreiranje instanci konkretnih klasa koje će biti popunjene podacima u zavisnosti od vrednosti koje su definisane za odgovarajući test fider.

Na slici 3.1.1 prikazan izgled CIM Profila koji je učitao u ovaj alat. Korišćenjem opcije Create RDF snippet, koja može da se koristi za konkretne klase, dobijamo šablon za kreiranje instance te klase sa svim njenim atributima i referencama, koji zatim jednostavno možemo nalepiti u .rdf fajl i popuniti prethodno navedenim podacima.



Slika 3.1.1 primer učitano CIM profila i korišćenje opcije Create RDF snippet



Na slici 3.1.2 prikazan je konačan deo izgleda RDF fajla koji sadrži instance i njihove vrednosti.

U prvom delu kao i kod CIM profila definisani su imenski prostori, a nakon toga koristi se CIM RDF vokabular za opis instanci, pa tako rdf:ID predstavlja identifikator resursa, a cim imenski prostor se koristi za identifikaciju klasa, atributa i reference koji su zatim popunjeni određenim vrednostima.

```
<?xml version="1.0" encoding="UTF-8"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:FTN="http://www.ftn.uns.ac.rs#"
  xmlns:cim="http://iec.ch/TC57/2010/CIM-schema-cim17#"
  xml:base="http://iec.ch/TC57/2010/CIM-schema-cim17#">
  <cim:ACLineSegment rdf:ID="_0b063747-9c83-4599-9541-80a39444ac3f">
    <cim:IdentifiedObject.mRID>0b063747-9c83-4599-9541-80a39444ac3f</cim:IdentifiedObject.mRID>
    <cim:IdentifiedObject.name>New Line</cim:IdentifiedObject.name>
    <cim:PowerSystemResource.PSRType rdf:resource="#Section"/>
    <cim:PowerSystemResource.AssetDatasheet rdf:resource="#e101cfa7-b2ba-4604-b79a-a3666da4c9ef_WI"/>
    <cim:Equipment.EquipmentContainer rdf:resource="#_8ac1fa62-a9c3-43cc-9bc8-3976d8b8e79b"/>
    <cim:ConductingEquipment.ratedVoltage>4800</cim:ConductingEquipment.ratedVoltage>
    <cim:Conductor.conductorType>Cable</cim:Conductor.conductorType>
    <cim:Conductor.length>243.84</cim:Conductor.length>
    <cim:ACLineSegment.PerLengthImpedance rdf:resource="#e101cfa7-b2ba-4604-b79a-a3666da4c9ef_PLI"/>
  </cim:ACLineSegment>
</rdf:RDF>
```

Slika 3.1.2 primer izgleda .rdf fajla sa ACLineSegment instancom, njenih atributima i referencama

Nakon definisanja ova dva dokumenta, oni su spremni za učitavanje u aplikaciju i dalji rad nad njima.

## 4.2 C# .NET APLIKACIJA ZA UČITAVANJE I SKLADIŠTENJE CIM MODELA I PRIMERA PODATAKA

### 4.2.1 KORISNIČKI INTERFEJS

Korisnički interfejs aplikacije je realizovan uz pomoć Windows Forms-a, UI frejmworka koji je deo .NET frejmworka.

Kreiranje korisničkog interfejsa svodi se na interakciju sa Toolbox-om Windows Forms-a.

Iz toolboxa bira se jedna od kontrola i jednostavnom drag & drop metodom prevuče se na radnu površinu aplikacije.

Moguće je menjati određene attribute ovih kontrola promenom vrednosti u okviru Properties prozora.

Neki od atributa koji se mogu menjati su veličina, font, naziv kontrole, kao i dizajn ime koje će se koristiti da bi se ova kontrola identifikovala u pozadinskom kodu koji služi da izvrši operaciju nakon korisničke interakcije sa kontrolom.

Na slici 3.1.3 moguće je videti prikaz korisničkog interfejsa aplikacije.

Prvi deo odnosi se na kreiranju kontrola koje omogućavaju korisniku biranje i učitavanje CIM modela odnosno RDF fajla instanciranih podataka.

Kontrole koje se odnose na Fuseki config omogućavaju ažuriranje konfiguracionog fajla Fuseki servera koji predstavlja RDF skladište podataka.

Kontrole u okviru RDF grafa omogućavaju pozivanje na neku od operacija manipulacija nad RDF grafom. RDF graf predstavlja strukturu podataka koja se koristi za čuvanje RDF trojki (subjekat – predikat – objekat).

Kontrole u okviru Query dela odnose se na pozivanje akcije postavljanja upita nad nekim iz combo box-a izabranim RDF grafom.

U delu Report u okviru RichTextBox-a biće ispisane rezultujuće vrednosti nastale izvršavanjem neke akcije, na primer klikom na List Graphs komandu u Report delu dobićemo ispis svih RDF grafova u okviru izabranog fuseki dataset combo box-a.

Za svaku od pomenutih kontrola se osluškuje neka akcija korisnika kao što je na primer klik na tu kontrolu, a onda se ta akcija obrađuje kao događaj u pozadinskom kodu pozivanjem callback funkcije koja ima za zadatak da dostavi korisniku odgovor u zavisnosti od toga koju je kontrolu izabrao.

| Select CIM Profile definition in RDFS/RDFS-Augmented format : |   |   |                             |
|---|---|---|-----------------------------|
| CIM Profile :   | <input type="text"/>                              |   | Browse..                    |
|   |   |   | Load Profile                |
| Fuseki config :   |   |   | Reset config                |
|   |   |   | Start Jena Storage          |
|   |   |   | Stop Jena Storage           |
| RDF/XML file :  | <input type="text"/>                              |   | Browse..                    |
|   |   |   | Load Data                   |
| Fuseki Dataset :  | <input type="text"/>                              |   | List Graphs                 |
| RDF Graph :   | <input type="text"/>                              |   | Load Graph                  |
|   | Subject (Qname) / Predicate (Qname) / Object :    | <input type="text"/>  | Update Graph                |
|   |   | <input checked="" type="checkbox"/> Add Triple <input type="checkbox"/> Delete Triple |                             |
|   |   |   | Delete Graph                |
|   |   |   | Save Graph                  |
| Query:  |   |   | Number of classes           |
|   | Class name ( leave empty field for all classes ): | <input type="text"/>  | Class-attribute information |
|   | Type of equipment:                                | <input type="text"/>  | Get equipment               |
|   | Enter component ID :                              | <input type="text"/>  | Get belonging equipment     |
|   | Enter ID:   | <input type="text"/>  | Get data                    |
| Report :  | <div></div>                                       |   |                             |

Slika 3.1.3 Korisnički interfejs aplikacije

#### 4.2.2 UČITAVANJE I PARSIRANJE CIM PROFILA I CIM PRIMERA PODATAKA

Uz pomoć dotNetRDF biblioteke, omogućeno je jednostavno rukovanje RDF fajlovima u okviru .NET okruženja. Funkcije koje obuhvata dotNetRDF na ovom nivou su čitanje, parsiranje, pisanje RDF fajlova.

Učitavanje CIM profila, na primer RDFS fajla i CIM primera podataka, na primer RDF fajla moguće je odraditi na dva načina:

- Parsiranjem modela direktno u Graph klasu
- Parsiranjem uz pomoć IRdfHandler

Za oba načina dotNetRDF nudi klase, metode i funkcije koje se pozivaju da bi se učitavanje i parsiranje dokumenta izvršilo.

Razlika između ova dva načina je u tome što prilikom direktnog parsiranja u Graph klasu, parsirani podaci su dostupni tek nakon celokupnog parsiranja dokumenta od početka do kraja, pa samim tim ako se koriste veliki dokumenti sa velikim brojem trojki to predstavlja vremenski neefikasno rešenje koje takođe oduzima veliku količinu memorije.

Drugi pristup koji je implementiran u okviru ove aplikacije, koristeći IRdfHandler, omogućava nam da rukujemo podacima čim je taj podatak parsiran, drugim rečima imamo kontrolu da odlučimo eksplicitno šta se dalje dešava sa nekim podatkom čim on postane parsiran.

IRdfHandler interfejs nudi metode koje se mogu modifikovati i tako prilagoditi potrebama ove aplikacije, ove metode su:

1. **StartRdfInternal()** – služi za inicijalizaciju podataka pred parsiranje, u ovom slučaju inicijalizaciju OntologyGraph i Graph klasa koje služe kao strukture podataka koje će prihvatiti parsirane trojke u zavisnosti od toga koji dokument, CIM profil ili CIM primer podataka se trenutno parsira
2. **HandleNamespacesInternal()** – služi da obradi slučaj kada se u toku parsiranja dođe do imenskog prostora. U zavisnosti od toga da li je dokument CIM profil ili CIM primer podataka,

imenski prostor sa svojim pridruženim prefiksom uz pomoć metode AddNamespace() će biti dodan u OntologyGraph ili Graph klasu

3. **HandleBaseUriInternal()** – opcioni korak koji služi za obradu osnovnog imenskog prostora, BaseURI
4. **HandleTripleInternal()** – služi da obradi slučaj kada se u toku parsiranja dođe do trojke, subjekat-predikat-objekat podatka, u ovom slučaju u zavisnosti od parsiranog dokumenta, trojka će biti parsirana i smeštena u okviru OntologyGraph-a ili Graph klasa
5. **EndRdfInternal()** – opcioni korak koji služi da se izvrše određene akcije nakon završetka parsiranja dokumenta kao što je na primer zatvaranje tog dokumenta

Ove metode će redom biti pozvane i proverene kada se dođe do određenog podatka u toku parsiranja.

Na slici 3.1.4 može se videti jedan isečak koda koji predstavlja osnovni deo procesa parsiranja RDF dokumenta u okviru ove aplikacije.

Kao ulazni parametri DoParse() funkcije prihvataju se pomenuti IRdfHandler objekat, bool success promenljiva koja predstavlja indikator o tome da li je parsiranje izvršeno uspešno i TimeSpan durationOfParsing promenljiva koja predstavlja indikator o ukupnom vremenu koje je bilo neophodno za parsiranje ovog dokumenta.

U okviru funkcije parsiranja koristi se try catch blok, try blok se prvi izvršava i u okviru njega koristimo gotov RdfXmlParser objekat i njegovu metodu Load() koja prihvata IRdfHandler objekat koji je implementiran u okviru RdfXmlParserHandler klase sa već pomenutim realizovanim metodama kao i filename objekat koji sadrži putanju do dokumenta koji se pokušava parsirati.

Ako u okviru parsiranja dođe do pojave greške (eng. Exception), promenljiva success će biti postavljen na false vrednost kao indikator neuspešnog parsiranja sa pridruženom porukom za korisnika, koja govori o tome gde i zašto je nastala greška.

U slučaju uspešnog parsiranja OntologyGraph i Graph klasa biće popunjene parsiranim trojkama a poruka o uspešnosti, vremenu parsiranja i prikazu parsiranih trojki će biti ispisana korisniku u okviru Report dela aplikacije.

```
1 reference
public class RdfXmlParser
{
    1 reference
    public static IRdfHandler DoParse(IRdfHandler handler, string filename, out bool success, out TimeSpan durationOfParsing)
    {
        success = true;
        durationOfParsing = new TimeSpan(0);

        DateTime startTime = DateTime.Now;
        DateTime stopTime;

        RdfXmlParserHandler handlerRdfXml = (RdfXmlParserHandler)handler;

        try
        {
            RdfXmlParser rdfParser = new RdfXmlParser();
            rdfParser.Load(handlerRdfXml, filename);
        }
        catch(RdfParseException parseEx)
        {
            success = false;
            throw new RdfParseException(parseEx.Message);
        }
        catch(RdfException rdfEx)
        {
            success = false;
            throw new RdfException(rdfEx.Message);
        }

        stopTime = DateTime.Now;

        durationOfParsing = stopTime - startTime;

        return handlerRdfXml;
    }
}
```

Slika 3.1.4 primer isečka koda u procesu parsiranja RDF dokumenta

## 4.2.3 FUSEKI RDF SKLADIŠTE I NJEGOVA SPREGA SA APLIKACIJOM UZ POMOĆ dotNETRDF BIBLIOTEKE

### 4.2.3.1 INICIJALIZACIJA KONFIGURACIONOG FAJLA FUSEKI SERVERA

RDF skladište predstavlja skladište koje omogućava manipulaciju nad RDF grafovima. U okviru aplikacije parsirani dokumenti su prvobitno čuvaju kao grafovi u okviru `OntologyGraph` ili `Graph` klasa, a zatim su posredstvom upotrebe `dotNetRDF` API sprege u interakciji sa RDF skladištem.

RDF skladište je obično realizovano kao neko od gotovih, javno dostupnih implementacija kao što su Apache Jena Fuseki, Sesame, Stardog, GraphDB itd.

Fuseki RDF skladište koje je deo ovog projekta izlaže pristupne tačke takozvane endpoint-ove preko HTTP protokola, kojima uz pomoć različitih funkcija ili metoda `dotNetRDF` biblioteke je moguće pristupiti i ostvariti komunikaciju sa ovim skladištem.

Pre samog pokretanja Fuseki skladišta, odnosno servera, neophodno je inicijalizovati konfiguracioni fajl `config.ttl` kojim se server postavlja u početno stanje poštujući pravila izložena u tom fajlu.

U ovom slučaju fuseki server je inicijalizovan na takav način da svaka posebna ontologija, koja je ustvari CIM profil koji se učitava i parsira predstavlja poseban fuseki dataset. Dataset se može zamisliti kao kontejner za RDF grafove, jedan dataset može da primi veliki broj grafova.

Razlog zbog kojeg se za svaku posebnu ontologiju kreira zaseban fuseki dataset stoji u tome da će nad svim RDF grafovima smeštenim u tim datasetovima biti primenjena ta ista ontologija i RDFS skup pravila za dodatno zaključivanje i pružanje dodatnih semantičkih podataka, koji mogu biti korisni kada se budu postavljali upiti nad ovim podacima.

Na slici 3.15 može da se vidi jedan primer `config.ttl` konfiguracionog fajla Fuseki servera.

```
PREFIX : <#>
PREFIX fuseki: <http://jena.apache.org/fuseki#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX ja: <http://jena.hpl.hp.com/2005/11/Assembler#>
PREFIX tdb: <http://jena.apache.org/2016/tdb#>

[] rdf:type fuseki:Server ;
   fuseki:services (
     :PowerTransformerService
   )
.

:PowerTransformerService rdf:type fuseki:Service ;
   fuseki:name "PowerTransformerServiceDataset" ;
   fuseki:endpoint [ fuseki:operation fuseki:query ] ;
   fuseki:endpoint [ fuseki:operation fuseki:update ] ;
   fuseki:endpoint [ fuseki:operation fuseki:gsp-rw ] ;
   fuseki:endpoint [
     fuseki:operation fuseki:query ;
     fuseki:name "sparql"
   ] ;
   fuseki:endpoint [
     fuseki:operation fuseki:query ;
     fuseki:name "query"
   ] ;
   fuseki:endpoint [
     fuseki:operation fuseki:update ;
     fuseki:name "update"
   ] ;
   fuseki:endpoint [
     fuseki:operation fuseki:gsp-r ;
     fuseki:name "get"
   ] ;
   fuseki:endpoint [
     fuseki:operation fuseki:gsp-rw ;
     fuseki:name "data"
   ] ;
   fuseki:dataset :rdfsDatasetPowerTransformerService ;
.

:rdfsDatasetPowerTransformerService rdf:type ja:RDFDataset ;
   ja:defaultGraph :inferenceModelPowerTransformer ;
.

:inferenceModelPowerTransformer rdf:type ja:InfModel ;
   ja:baseModel :PowerTransformerGraph ;
   ja:reasoner [
     ja:reasonerURL <http://jena.hpl.hp.com/2003/RDFSExpRuleReasoner>
   ]
.

:PowerTransformerGraph rdf:type ja:MemoryModel ;
   ja:content [ ja:externalContent <d:/nebojsa_knezevic_projekat_pr_104_2015/testfiles/powertransformer.legacy-rdfs> ] ;
.
```

Slika 3.1.5 primer `config.ttl` fajla Fuseki servera

Prvi deo odnosi se na definisanje svih neophodnih prefiksa imenskih prostora čiji vokabular će se koristiti prilikom definisanja ove konfiguracije.

Nakon toga u okviru fuseki:services bloka navode se svi servisi, odnosno datasetovi koji će biti izloženi putem HTTP-a korisnicima kako bi se omogućilo postavljanje SPARQL upita nad grafovima u okviru ovih servisa.

Svaki servis se pored toga što se navodi dodatno i definiše skupom mogućih operacija koje je moguće izvršiti putem pristupnih tačaka ovog servisa.

Najčešće korišćene operacije su:

- **fuseki:query** – omogućava postavljanje upita nad grafovima datog dataseta
- **fuseki:update** – omogućava modifikaciju grafova, obično njihovih trojki datog dataseta
- **fuseki:gsp-rw** – odnosi se na Graph Store Protocol, omogućava izmene nad grafovima, bitan jer je neophodan za API sprega sa dotNetRDF bibliotekom

Svako od ovih pristupnih tačaka moguće je uz pomoć **fuseki:name** parametra pridružiti proizvoljno ime, to ime će biti deo URL adrese koja identifikuje koji servis je pozvan.

Dataset je definisan uz pomoć **fuseki:dataset** parametra, u okviru ove konfiguracije definisan je kao memorijski model, čiji podaci nisu perzistentni prilikom gašenja Fuseki servera, što označava **ja:RDFDataset** parametar.

U okviru dataseta definiše se i model zaključivanja kao kombinacija CIM profila odnosno ontologije podataka i specijalnog Jena RDFS Reasonera na osnovu kojeg se dobija dodatna semantika, odnosno značenja podataka.

Inicijalizacija config.ttl fajla u okviru aplikacije poziva se klikom na **Apply loaded RDFS ontology over fuseki dataset** dugme, nakon kojeg se koristi StreamWriter C# klasa za otvaranje i modifikovanje ovog fajla.

#### 4.2.3.2 STARTOVANJE FUSEKI SERVERA

Odnosi se na pokretanje Fuseki server instance na lokalnoj mašini, a realizuje se korišćenjem Process klase u okviru C# .NET-a, primer koda može se videti na slici 3.1.6.

```
1 reference
public void StartFusekiStore()
{
    Process jenaStorage = new Process();
    jenaStorage = HelperMethods.InitializeJenaStorageProcess(jenaStorage);

    string command = "fuseki-server";
    jenaStorage.StartInfo.Arguments = "/k " + command;

    jenaStorage.Start();

    File.WriteAllText("jenaProcessId.txt", jenaStorage.Id.ToString());
}
```

Slika 3.1.6 primer koda koji se koristi za pokretanje instance Fuseki servera

Pre samog izvršavanja pokretanja servera, neophodno je u okviru Process klase definisani početne argumente. U InitializeJenaStorageProcess() se podešava putanja do radnog direktorijuma iz kojeg se komanda **/k fuseki-server** izvršava.

Ova komanda predstavlja komandu koju definiše sam fuseki-server.bat fajl koji se dobija preuzimanjem Fuseki servera sa apache jena veb sajta, a koja se na ovaj način poziva iz C# koda.

Nakon izvršavanja ovog koda pokreće se cmd.exe koji izlistava povratne informacije korisniku o fuseki serveru, koji mogu da se vide na slici 3.1.7. Server se na lokalnoj mašini pokreće na adresi <http://localhost:3030/>.

```
C:\WINDOWS\SYSTEM32\cmd.exe - fuseki-server
18:14:20 INFO Server      :: Apache Jena Fuseki 4.8.0
18:14:20 INFO Config      :: FUSEKI_HOME=C:\apache-jena-fuseki-4.8.0\apache-jena-fuseki-4.8.0\
18:14:20 INFO Config      :: FUSEKI_BASE=C:\apache-jena-fuseki-4.8.0\apache-jena-fuseki-4.8.0\run
18:14:20 INFO Config      :: Shiro file: file:///C:\apache-jena-fuseki-4.8.0\apache-jena-fuseki-4.8.0\run
18:14:22 INFO Server      :: Configuration file: C:\apache-jena-fuseki-4.8.0\apache-jena-fuseki-4.8.0\run
18:14:22 INFO Server      :: Path = /PowerTransformerServiceDataset
18:14:22 INFO Server      :: Memory: 1.2 GiB
18:14:22 INFO Server      :: Java: 20.0.1
18:14:22 INFO Server      :: OS: Windows 10 10.0 amd64
18:14:22 INFO Server      :: PID: 8976
18:14:22 INFO Server      :: Started 2023/08/04 18:14:22 UTC on port 3030
```

Slika 3.1.7 informacije o pokrenutom fuseki serveru prikazane u okviru cmd.exe

Odmah posle startovnja Fuseki server koristeći Fuseki HTTP Administration Protocol Fuseki serveru se šalje poruka da dostavi sve pokrenute datasetove kako bi one mogle biti priložene korisniku u okviru Fuseki datasets combo box-a. Primer koda nalazi se na slici 3.1.8.

```
1 reference
public List<string> GetAllDatasetsFromFuseki(ref string errorMessage, ref string datasetInfo)
{
    string getDatasetsURL = "http://localhost:3030/$/datasets";

    string parsedJSON = string.Empty;

    List<string> datasetList = new List<string>();
    JSONArray datasetsListJSON = new JSONArray();

    using (HttpClient client = new HttpClient())
    {
        try
        {
            HttpResponseMessage response = client.GetAsync(getDatasetsURL).Result;
            if (response.IsSuccessStatusCode)
            {
                string responseData = response.Content.ReadAsStringAsync().Result;
                datasetList = HelperMethods.ExtractDatasets(responseData, ref datasetInfo);
                //parsedJSON = HelperMethods.JSON(datasetsListJSON);
            }
            else
            {
                errorMessage = $"Error: {response.StatusCode} - {response.ReasonPhrase}";
            }
        }
        catch (HttpRequestException ex)
        {
            errorMessage = $"HttpRequestException occurred: {ex.Message}";
        }
        catch (InvalidOperationException ex)
        {
            errorMessage = $"InvalidOperationException occurred: {ex.Message}";
        }
        catch (Exception ex)
        {
            errorMessage = $"Exception occurred: {ex.Message}";
        }
    }

    return datasetList;
}
```

Slika 3.1.8 prikaz koda koji dostavlja sve inicijalizovane Fuseki datasetove sa Fuseki servera

Funkcija GetAllDatasetsFromFuseki() sadrži dva parametra errorMessage i datasetInfo, prvo se postavlja URL kojim od Fuseki servera zahtevamo datasetove, a koji se postavlja tako što se adresi na kojoj je startovan Fuseki server doda znak \$ koji označava da se radi o Fuseki HTTP Administration Protocol-u, a potom i datasets koji označava da želimo da u rezultatu dobijemo sve trenutno pokrenute datasetove.

Koristeći HttpClient klasu na osnovu URL adrese pristupa se Fuseki serveru od kojeg zatim uz pomoć metode GetAsync() i svojstva Result tražimo sve datasetove i smeštamo ih u listu i dostavljamo combo box-u na UI.

#### 4.2.3.3 OPERACIJE MANIPULACIJE NAD RDF GRAFOVIMA U SPREZI SA FUSEKI SERVEROM

Operacije manipulacije nad RDF grafovima odnose se na korišćenje triple store integration dela dotNetRDF biblioteke, koja definiše sledeće operacije:

- **ListGraphs()** – metoda omogućava dobavljanje svih RDF grafova u odnosu na izabrani Fuseki dataset
- **LoadGraph()** – metoda omogućava učitavanje RDF grafa iz Fuseki RDF skladišta u Graph klasu, a zatim i prikaz trojki koji čine taj graf korisniku
- **UpdateGraph()** – metoda omogućava modifikaciju trojki izabranog RDF grafa u okviru nekog Fuseki dataseta
- **SaveGraph()** – metoda omogućava skladištenje RDF grafa u Fuseki RDF skladište u okviru nekog Fuseki dataseta
- **DeleteGraph()** – metoda omogućava brisanje RDF grafa iz nekog Fuseki dataseta

Prilikom poziva ovih metoda, pre njihovog samog izvršavanja pozivaju se funkcije koje proveravaju da li je Fuseki server online i funkcija konektovanja na pristupnu tačku Fuseki servera.

Na slici 3.1.9 može da se vidi funkcija provere dostupnosti Fuseki servera.

```
public bool IsFusekiOnline(ref string responseMessage)
{
    string pingFusekiUrl = "http://localhost:3030/$/ping";

    using (HttpClient client = new HttpClient())
    {
        try
        {
            HttpResponseMessage response = client.GetAsync(pingFusekiUrl).Result;
            if (response.IsSuccessStatusCode)
            {
                string pingResponse = response.Content.ReadAsStringAsync().Result;
                DateTimeOffset pingTimestamp = DateTimeOffset.Parse(pingResponse);
                return true;
            }
            else
            {
                responseMessage = $"Error: {response.StatusCode} - {response.ReasonPhrase}";
                return false;
            }
        }
        catch (HttpRequestException ex)
        {
            responseMessage = $"HttpRequestException occurred: {ex.Message}";
            return false;
        }
        catch (InvalidOperationException ex)
        {
            responseMessage = $"InvalidOperationException occurred: {ex.Message}";
            return false;
        }
        catch (Exception ex)
        {
            responseMessage = $"Exception occurred: {ex.Message}";
            return false;
        }
    }
}
```

Slika 3.1.9 funkcija provere dostupnosti Fuseki servera

Ova funkcija se oslanja na Fuseki HTTP Administration Protocol, koja definiše URL uz pomoć kojeg Fuseki server vraća određeni HTTP Status kod, ako je ovaj kod uspešan, dalje može da se nastavi sa pristupom na Fuseki server, u suprotnom korisnik biva obavešten da trenutno nije moguće pristupiti serveru.

U slučaju da može da se pristupi Fuseki serveru poziva se funkcija sa slike 3.2.1.

Promenljiva fuseki se u zavisnosti od toga da li postoji već neka konekcija sa pristupnom tačkom Fuseki servera inicijalizovana klasom FusekiConnector koja je definisana kao klasa za pristup Fuseki serveru u okviru dotNetRDF biblioteke i konektuje se na izabrani fuseki dataset koji je korisnik izabrao iz combo box-a iz korisničkog interfejsa.

Dodavanje dela /data označava da će dotNetRDF operacije koristiti Graph Store protocol radi komunikacije sa Fuseki servisima.

```

10 references
public void ConnectToDatasetEndpoint(string selectedDataset)
{
    if (fuseki == null)
    {
        fuseki = new FusekiConnector(baseServiceUri + selectedDataset + "/data");
    }
    else
    {
        string currentDatasetConnected = HelperMethods.ExtractDataset(fuseki.ToString());

        if (!currentDatasetConnected.Equals(selectedDataset))
        {
            fuseki = null;
            fuseki = new FusekiConnector("http://localhost:3030/" + selectedDataset + "/data");
        }
    }
}

```

Slika 3.2.1 funkcija konektovanja na pristupu tački Fuseki servera i određeni fuseki dataset

Izvršavanje funkcije ListGraphs() i LoadGraph(Graph g, string graphUri) je trivijalno i svodi se na jednostavno pozivanje funkcije u slučaju ListGraphs(), dok je u slučaju funkcije LoadGraph potrebno proslediti instancu Graph klase u koju će biti učitani graf iz Fuseki RDF skladišta i naziv grafa koji je korisnik izabrao iz combo box sa UI.

Prilikom izvršavanja funkcija DeleteGraph(string graphUri) i SaveGraph(Graph g) osim prosleđivanja imena grafa odnosno Graph klase neophodno je ažurirati i default graf fuseki dataseta u okviru kojeg se ovi grafovi nalaze.

Default graph u okviru sadrži ontologiju (trojke CIM profila) i RDFS pravila definisana Jena RDFS Reasoner-om koji definišu semantiku ovih grafova, pa samim tim prilikom brisanja ili skladištenja grafa potrebno je i obrisati odnosno dodati trojke tog grafa iz default grafa, to je prikazano na slici 3.2.2 za slučaj izvršavanja DeleteGraph() metode.

```

1 reference
public bool DeleteGraph(string selectedDataset, string graphName)
{
    bool deleted = false;
    Graph loadedGraph = new Graph();
    ConnectToDatasetEndpoint(selectedDataset);

    if (fuseki.DeleteSupported)
    {
        fuseki.LoadGraph(loadedGraph, graphURIs[graphName] + graphName);
        fuseki.UpdateGraph(string.Empty, null, loadedGraph.Triples);

        fuseki.DeleteGraph(baseURI + graphName);
        deleted = true;
        graphURIs.Remove(graphName);
    }
    else
    {
        deleted = false;
    }

    return deleted;
}

```

Slika 3.2.2 prikaz DeleteGraph() metode za brisanje grafa

Da bi se prilikom brisanja grafa iz Fuseki RDF skladišta obrisale i trojke u okviru default grafa, prvo se učitava traženi graf u Graph instancu loadedGraph a zatim uz pomoć UpdateGraph() metode modifikujemo default graf tako što mu prosleđujemo trojke traženog grafa na brisanje.

Nakon ovoga preostaje još samo brisanje traženog grafa iz RDF skladišta.



#### 4.2.3.4 OPERACIJE POSTAVLJANJA UPITA NAD RDF GRAFOVIMA U SPREZI SA FUSEKI SERVEROM KORISTEĆI SPARQL

Postavljanje upita nad CIM instanciranim primerom podacima u odnosu na ontologiju (CIM profil) i RDFS Reasoner omogućava dobijanje određenih informacija o stanju, opremi i vrednostima distributivne mreže test fidera.

U okviru ove aplikacije omogućeno je postavljanje sledećih upita:

- Broj svih klasa
- Dobijanje informacija o prekidačkoj ili provodnoj opremi
- Dobijanje informacija o opremi koje pripada nekoj komponenti ili podstanici
- Dobijanje informacija o komponenti u odnosu na unet ID te komponente

Upiti nad RDF skladište se postavljaju putem SPARQL jezika koji je analogan SQL jeziku koji se koristi za postavljanje upita na relacionim bazama podataka.

Biblioteka dotNetRDF sadrži specijalne klase i metode koje omogućavaju postavljanje SPARQL upita.

Korisnik bira Fuseki dataset i RDF graf u okviru tog dataseta nad kojim želi da se postavi upit, a zatim se to prosleđuje funkciji kao parametar.

Klasa `SparqlParametrizedString` pruža mogućnost kreiranja SPARQL upita korišćenjem SPARQL vokabulara. Jedan primer kreiranja SPARQL upita može se videti na slici 3.2.3 prilikom postavljanja upita o provodnoj ili prekidačkoj opremi koja je sadržana u test fideru.

```
public SparqlResultSet GetTypeOfEquipment(string selectedDataset, string graphName, string typeOfEquipment)
{
    ConnectToDatasetEndpoint(selectedDataset);

    Graph selectedGraph = LoadGraph(selectedDataset, graphName);
    SparqlQueryParser sparqlParser = new SparqlQueryParser();
    SparqlParameterizedString cmdString = new SparqlParameterizedString();

    cmdString.Namespaces = selectedGraph.NamespaceMap;
    cmdString.QueryProcessor = GetQueryProcessor();
    cmdString.CommandText = "SELECT DISTINCT ?class ?property ?value " +
        "WHERE { " +
        "  ?class rdf:type @value . " +
        "  ?class ?property ?value " +
        "  FILTER (?value != rdfs:Resource) " +
        "} ";

    if (typeOfEquipment == "Switch")
    {
        cmdString.SetUri("value", new Uri(graphURIs[graphName] + "Switch"));
    }
    else if (typeOfEquipment == "Conducting Equipment")
    {
        cmdString.SetUri("value", new Uri(graphURIs[graphName] + "ConductingEquipment"));
    }

    SparqlQuery query = sparqlParser.ParseFromString(cmdString);
    Object results = cmdString.QueryProcessor.ProcessQuery(query);

    if (results is SparqlResultSet)
    {
        SparqlResultSet result = (SparqlResultSet)results;
        return result;
    }
    else
    {
        return null;
    }
}
```

Slika 3.2.3 primer postavljanja SPARQL upita nad RDF skladištem

Nakon konektovanja na pristupnu tačku Fuseki servera `selectedGraph` promenljiva se postavlja na vrednost grafa koji je putem UI korisnik tražio putem metode `LoadGraph`.

Sledeći korak predstavlja popunjavanje instance klase `SparqlParametrizedString` podacima kao što su neophodni imenski prostori koji su neophodni za korišćenje određenog vokabulara kao što je na primer `rdf` imenski prostor i ključna reč `type` (`rdf:type`).

Nakon toga sledi inicijalizacija SPARQL procesora koji omogućava postavljanje upita nad Fuseki skladištem, u ovom slučaju to je klasa `GenericQueryProcessor`.

Glavni parametar koji definiše sam upit je `CommandText`. On se popunjava stringom, gde se u `SELECT` delu navodi koje promenljive je potrebno vratiti kao odgovor. To može biti skup podataka ili prazan skup u slučaju da Fuseki server nema podatak koji odgovara uslovu upita.

`DISTINCT` deo označava da će biti uzeti u obzir samo jedinstvena rešenja.

`WHERE` blok definiše uslov upita. Na osnovu ovog uslova upit će biti izvršen i biće vraćena samo rešenja koja zadovoljavaju definisani uslov.

U prezentovanom slučaju na slici 3.2.3 traže se rešenja koja će biti smeštena u promenljive `class`, `property`, `value` trojku. SPARQL definiše promenljivu tako što se ispred imena promenljive nalazi `?` znak.

U okviru `WHERE` uslovnog bloka traže se svi subjekti koji su tipa (`rdf:type`) `@value`, a `value` je u zavisnosti od korisnikovog izbora sa `UI`, `Switch` ili `ConductingEquipment` URI, u redu ispod navodi se dalje da za sve takve subjekte koje odgovaraju ovom uslovu izlistaju dodatno i njihov predikat i objekat i to vrati kao povratna vrednost.

Nakon definisanja upita, upit se parsira uz pomoć metode `ParseFromString` u `SparqlQuery` objekat i zatim nad tim objektom uz pomoć prethodno pomenutog procesora se izvršava SPARQL upit.

Fuseki server procesira upit i vraća kao povratnu vrednost odgovor koji je tipa `SparqlResultSet` odnosno to je `dotNetRDF` klasa koja sadrži skup svih rešenja.

Analogno ovom primeru izvršava se na potpuni isti način i ostali upiti s tim da jedina izmena predstavlja samo vrednost `CommandText` parametra.

#### 4.2.3.5 PERFORMANSE REŠENJA

Aplikacija je pravljenja sa ciljom testiranje malih distributivnih izvoda i kao takva izuzetno je efikasna i brza prilikom rada sa malim CIM profilima i primerima podataka.

Brzina parsiranja ovakvih dokumenata može da se vidi na slici 3.2.4 koja pokazuje izuzetno brzo učitavanje i parsiranje kada su dokumenti veličine eksperimentalnih distributivnih mreža, odnosno test fidera

```
RDF file loaded:
Duration of RDF file loading: 00:00:00.0947457
loading RDF File was successful

-----Done parsing profile-----
```

Slika 3.2.4 brzina parsiranja RDF dokumenta u okviru priložene aplikacije

Sa ovim na umu inicijalizovano je i Fuseki RDF skladište kao memorijski model koji lako i brzo može da omogući skladištenje i postavljanje upita nad podacima.

### 4.3 C# .NET APLIKACIJA ZA POTREBE TESTIRANJA PERFORMANSI JENA RDF SKLADIŠTA I RELACIONIH BAZA

#### 4.3.1 TESTIRANJE PERFORMANSI JENA RDF SKLADIŠTA

Testiranje performansi je realizovano uz pomoć `BenchmarkDotNet` biblioteke i ono se može podeliti na 2 dela:

- Inicijalizaciju neophodnih objekata pre neposrednog izvršavanja merenja
- Merenje performansi CRUD operacija i SPARQL upita prilikom interakcije sa Jena RDF skladištem

#### 4.3.1.1 INICIJALIZACIJA OBJEKATA PRE NEPOSREDNOG IZVRŠAVANJA MERENJA

U okviru inicijalizacije neophodnih objekata u slučaju rada sa Jena RDF skladištem BenchmarkDotNet biblioteka nudi poseban atribut [Global Setup], koji se koristi da se obeleži funkcija koja sadrži određenu logiku koja je potrebna da se izvrši jednom pre neposrednog merenja, a za tu samu logiku se ne vrši merenje performansi.

U okviru gore navedene funkcije inicijalizuju se sledeći objekti:

- Posebni konektori ka in-memory datasetu i tdb2 datasetu Fuseki servera, koristeći FusekiConnector klasu dotNetRDF biblioteke i dozvoljeno vreme HTTP timeouta, odnosno nakon koliko milisekundi će HTTP veza sa Fuseki serverom biti prekinuta ako slanje ili primanje poruke nije izvršeno
- Dictionary<string, IGraph> objekat koji sadrži RDF fajlove različitih veličina i broja RDF trojki koji će se koristiti za potrebe merenja performansi
- IGraph objekat koji sadrži RDF trojke koje će se koristiti u okviru testiranja Update operacije
- SparqlQueryParser, SparqlParametrizedString i GenericQueryProcessor objekti koji će se koristiti u svrhu formiranja SPARQL upita koji će se dalje slati Jena RDF skladištu

Na slici 3.2.5 prikazana je pomenuta funkcija u okviru koje se vrši gore navedeno inicijalizovanje objekata

```
[GlobalSetup]
0 references
public void Setup()
{
    inMemoryConnector = new FusekiConnector(baseServiceUri + "FTNServiceDataset" + "/data");
    inMemoryConnector.Timeout = 100000;
    tdb2Connector = new FusekiConnector(baseServiceUri + "TDB2ServiceDataset" + "/data");
    tdb2Connector.Timeout = 100000;
    testGraphs = new Dictionary<string, IGraph>
    {
        {"small", handler.LoadRDF("D:\\Webojsa_Knezevic_Projekat_PR_104_2015\\TestFiles\\modelLabs_PowerTransformer_Example.xml") },
        {"medium", handler.LoadRDF("D:\\Webojsa_Knezevic_Projekat_PR_104_2015\\TestFiles\\CGMES_v2.4.15_RealGridTestConfiguration_SSH_V2.xml") },
        {"large", handler.LoadRDF("D:\\Webojsa_Knezevic_Projekat_PR_104_2015\\TestFiles\\CGMES_v2.4.15_RealGridTestConfiguration_EQ_V2.xml") }
    };

    updateGraphTest = handler.LoadRDF("D:\\Webojsa_Knezevic_Projekat_PR_104_2015\\TestFiles\\20171002T0930Z_1D_IL_TP_2.xml");
    sparqlQueryParser = new SparqlQueryParser();
    cmdString = new SparqlParameterizedString();

    processor = new GenericQueryProcessor(inMemoryConnector);
    processorTDB2 = new GenericQueryProcessor(tdb2Connector);

    cmdString.QueryProcessor = processor;
    inMemoryConnector.UpdateGraph(string.Empty, testGraphs["small"].Triples, null);
}
```

Slika 3.2.5 Primer izgleda Global Setup funkcije za slučaj Jena RDF skladišta

#### 4.3.1.2 MERENJE PERFORMANSI CRUD OPERACIJA I SPARQL UPITA PRILIKOM INTERAKCIJE SA JENA RDF SKLADIŠTEM

Merenje se izvršava tako što se funkcijama koje želimo da testiramo dodeljuje [Benchmark] atribut BenchmarkDotNet biblioteke, a potom uz pomoću metode BenchmarkRunner.Run<T>(), koja se poziva iz main funkcije započinje merenje.

U slučaju CRUD operacija za in-memory i tdb2 jena skladišta testiraju se prethodno pomenute operacije dotNetRDF biblioteke:

- SaveGraph()
- UpdateGraph()
- LoadGraph()
- DeleteGraph()

Primer testiranja operacije SaveGraph(), koja skladišti RDF graf u okviru memorije Fuseki servera u slučaju in-memory dataset ili na disku u slučaju tdb2 dataseta je prikazan na slici 3.2.6.

Atribut [Arguments] omogućava da se priloži određen skup vrednosti koji mora biti konstanta i koji će se prilikom kompajliranja mapirati na parametre SaveGraph() funkcije. Svrha korišćenja ovog atributa je da se vide promene u performansama u zavisnosti od različitih ulaznih podataka, a to su u ovom slučaju RDF fajlovi različitih veličina, odnosno broj RDF trojki, takođe podešava se i baseUri parametar kako bi Fuseki server na osnovu tog parametra dodelio ime RDF grafu koji se skladišti u okviru dataseta.

In-memoryConnector i tdb2Connector, prethodno inicijalizovani u okviru Setup funkcije (vidi sliku 3.2.5), koriste se zatim da bi se pozvala SaveGraph() operacija nad in-memory, odnosno tdb2 datasetom.

```
[Benchmark]
[Arguments("http://iec.ch/TC57/2013/CIM-schema-cim15#", "modelLabs_PowerTransformer_Example.xml")]
[Arguments("http://iec.ch/TC57/2013/CIM-schema-cim16#", "CGMES_v2.4.15_RealGridTestConfiguration_SSH_V2")]
[Arguments("http://iec.ch/TC57/2013/CIM-schema-cim16#", "CGMES_v2.4.15_RealGridTestConfiguration_EQ_V2")]
0 references
public void SaveGraph_InMemory(string baseUri, string graphName)
{
    IGraph graph = GetGraph(graphName);

    graph.BaseUri = new Uri(baseUri + graphName);

    inMemoryConnector.SaveGraph(graph);
}

[Benchmark]
[Arguments("http://iec.ch/TC57/2013/CIM-schema-cim15#", "modelLabs_PowerTransformer_Example.xml")]
[Arguments("http://iec.ch/TC57/2013/CIM-schema-cim16#", "CGMES_v2.4.15_RealGridTestConfiguration_SSH_V2")]
[Arguments("http://iec.ch/TC57/2013/CIM-schema-cim16#", "CGMES_v2.4.15_RealGridTestConfiguration_EQ_V2")]
0 references
public void SaveGraph_TDB2(string baseUri, string graphName)
{
    IGraph graph = GetGraph(graphName);

    graph.BaseUri = new Uri(baseUri + graphName);

    tdb2Connector.SaveGraph(graph);
}
```

Slika 3.2.6 primer testiranja SaveGraph() funkcije u okviru in-memory i tdb2 dataseta Jena RDF skladišta

Analogno testiranju SaveGraph() operacije se testiraju i operacije UpdateGraph(), LoadGraph() i DeleteGraph()

U okviru testiranja SPARQL upita testiraju se dva slučaja:

- Testiranje performansi jednostavnog(trivijalnog) SPARQL upita
- Testiranje performansi kompleksnog SPARQL upita

Primer testiranja jednostavnog SPARQL upita prikazan je na slici 3.2.7.

Ponovo kao i u slučaju testiranja CRUD operacija funkcija SparqlSimpleQuerySmallRDF\_InMemory se obeležava atributima [Benchmark] i [Arguments].

Potom se koristi SparqlParametrizedString objekat koji je inicijalizovan u okviru Setup funkcije, a koji uz pomoć svog svojstva CommandText prihvata tekst SPARQL upita predstavljen stringom.

Nakon definisanja SPARQL upita, kako bi se upit mogao proslediti Jena Fuseki serveru i izvršiti nad željenim datasetom potrebno je postaviti QueryProcessor properti.

QueryProcessor properti prihvata GenericQueryProcessor objekat koji je inicijalizovan u Setup funkciji, a koji omogućava postavljanje SPARQL upita nad svakim skladištem koje implementira interfejs IQueryStorage u okviru dotNetRDF biblioteke, u ovom slučaju GenericQueryProcessor je inicijalizovan uz pomoć InMemoryConnector objekta kako bi SPARQL upit bio izvršavan nad In-memory datasetom.

SparqlQuery klasa se koristi kako bi predstavila jedan SPARQL upit u strukturiranom obliku, da bi se dobio SparqlQuery objekat SPARQL upit, koji je do sada bio predstavljen stringom parsira se uz pomoć ParseFromString metodom.

Konačno ProcessQuery metoda prihvata SparqlQuery objekat i izvršava SPARQL upit nad datasetom Jena Fuseki RDF skladišta na osnovu podataka u okviru GenericQueryProcessor objekta.

SparqlResultSet sadrži tabelu rezultata koji Jena Fuseki skladište vraća kao odgovor na SELECT upit, svakom individualnom rezultatu moguće je pristupiti iterirajući kroz ovu tabelu (skup).

Analogno testiranju jednostavnog SPARQL upita za in-memory skladišta testira se i za tdb2 skladište, pri čemu je samo potrebno QueryProcessor svojstvu cmdString objekta SparqlParametrizedString klase dodeliti GenericQueryProcessor, koji je inicijalizovan u Setup funkciji koristeći tdb2Connector objekat.

```
[Benchmark]
[Arguments("http://iec.ch/TC57/2013/CIM-schema-cim15#", "modellabs_PowerTransformer_Example.xml")]
0 references
public SparqlResultSet SparqlSimpleQuerySmallRDF_InMemory(string baseUri, string graphName)
{
    cmdString.CommandText = "PREFIX cim: < http://iec.ch/TC57/2010/CIM-schema-cim15#> " +
        "SELECT ?baseVoltage ?nominalVoltage " +
        "FROM <" + baseUri + graphName + "> " +
        "WHERE { " +
            "?baseVoltage a cim:BaseVoltage . " +
            "?baseVoltage cim:BaseVoltage.nominalVoltage ?nominalVoltage . " +
        "}";

    cmdString.QueryProcessor = processorInMemory;
    SparqlQuery query = sparqlQueryParser.ParseFromString(cmdString);
    Object results = cmdString.QueryProcessor.ProcessQuery(query);

    if (results is SparqlResultSet)
    {
        SparqlResultSet result = (SparqlResultSet)results;
        return result;
    }
    else
    {
        return null;
    }
}
```

Slika 3.2.7 primer funkcije testiranja jednostavnog SPARQL upita u odnosu na in-memory dataset Jena RDF skladišta

Testiranje performansi kompleksnog SPARQL upita vrši se na isti način kao u slučaju jednostavnog SPARQL upita s jedinom razlikom koja je sama komanda, primer kompleksnog SPARQL upita za tdb2 skladište prikazan je na slici 3.2.8.

```
[Benchmark]
[Arguments("http://iec.ch/TC57/2013/CIM-schema-cim15#", "modellabs_PowerTransformer_Example.xml")]
0 references
public SparqlResultSet SparqlComplexQuerySmallRDF_TDB2(string baseUri, string graphName)
{
    cmdString.CommandText = "PREFIX cims:<http://iec.ch/TC57/1999/rdf-schema-extensions-19990926#> " +
        "PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#> " +
        "PREFIX rdf:< http://www.w3.org/1999/02/22-rdf-syntax-ns#> " +
        "SELECT DISTINCT ?anySubClass ?property ?type " +
        "WHERE { " +
            "{ " +
                "?anySubClass rdfs:subClassOf ?anyClass . " +
                "?property rdfs:domain ?anyClass . " +
                "?property cims:datatype| rdfs:range ?type " +
                "FILTER (ISTRSTARTS(STR(?anySubClass), str(rdf:)) && ISTRSTARTS(STR(?anySubClass), str(rdfs:))) " +
            } " +
            "UNION " +
            "{ " +
                "?anySubClass rdf:type rdfs:Class . " +
                "?anySubClass ?property ?type " +
                "FILTER NOT EXISTS( ?anySubClass cims:belongsToCategory []) " +
                "FILTER (ISTRSTARTS(STR(?anySubClass), str(rdf:)) && ISTRSTARTS(STR(?anySubClass), str(rdfs:)) && ISTRSTARTS(STR(?anySubClass), str(cims:ClassCategory))) " +
                "&& ?anySubClass != ?type && ?type != rdfs:Resource && ?property != rdfs:comment && ?property != rdfs:label " +
            } " +
            "UNION " +
            "{ " +
                "?anySubClass rdf:type rdfs:Class . " +
                "?property rdf:type ?anySubClass . " +
                "?property rdfs:label ?label . " +
                "?property rdfs:type ?type " +
                "FILTER (ISTRSTARTS(STR(?anySubClass), str(rdf:)) && ISTRSTARTS(STR(?anySubClass), str(rdfs:)) && ISTRSTARTS(STR(?anySubClass), str(cims:ClassCategory))) " +
            } " +
        " " +
        "ORDER BY ?anySubClass";

    cmdString.QueryProcessor = processorTDB2;
    SparqlQuery query = sparqlQueryParser.ParseFromString(cmdString);
    Object results = cmdString.QueryProcessor.ProcessQuery(query);

    if (results is SparqlResultSet)
    {
        SparqlResultSet result = (SparqlResultSet)results;
        return result;
    }
    else
    {
        return null;
    }
}
```

Slika 3.2.8 primer funkcije testiranja kompleksnog SPARQL upita u odnosu na tdb2 dataset Jena RDF skladišta

### 4.3.2 TESTIRANJE PERFORMANSI RELACIONIH BAZA PODATAKA

Merenje performansi relacionih baza odnosi se na testiranje performansi MySQL i PostgreSQL baza podataka.

Pre samog početka testiranja performansi uz pomoć BenchmarkDotNet [11] biblioteke neophodno je obezbediti sledeće korake kako bi se uspostavila komunikacija, uz pomoć Entity Framework 6, između .NET aplikacije, MySQL i PostgreSQL baze podataka:

- Kreiranje data modela
- Kreiranje DbContext klase
- Konfiguracija migracija

Kreiranje data modela u okviru code-first approach Entity Framework-a se odnosi na kreiranje entiteta koji se nakon toga mapiraju u bazi podataka na tabele, dok se atributi(property) entiteta mapiraju na kolone tih tabela.

Primer data modela korišćen u okviru ove aplikacije prikazan je na slici 3.2.9 i 3.3. Na slici 3.2.9 prikazan je način modelovanja jednog RDF graf entiteta, odnosno jednog celokupnog RDF fajla sa svim svojim pridruženim trojkama, dok se na slici 3.3 vidi modelovanje svakog pojedinačnog RDF triple entiteta, odnosno svake trojke subjekat-predikat-objekat koja pripada nekog RDF grafu.

```
4 references
public class RDFGraph
{
    2 references
    public int RDFGraphId { get; set; }
    4 references
    public string GraphName { get; set; }
    2 references
    public List<RDFTriple> Triples { get; set; }
}
```

Slika 3.2.9 primer modelovanja jednog RDF graf entiteta

```
13 references
public class RDFTriple
{
    4 references
    public int RDFTripleId { get; set; }
    2 references
    public int RDFGraphId { get; set; }
    6 references
    public string Subject { get; set; }
    6 references
    public string Predicate { get; set; }
    6 references
    public string Object { get; set; }
    2 references
    public string GraphName { get; set; }
}
```

Slika 3.3 primer modelovanja jednog RDF triple entiteta

RDF graf entitet kao primarni ključ u okviru tabele baze podataka sadrži RDFGraphId, odnosno jedinstveni broj koji se dodeljuje svakom posebnom RDF grafu koji se skladišti u okviru MySQL ili PostgreSQL baze podataka.

GraphName predstavlja ime koje je dodeljeno unetom RDF grafu, dok Triples predstavlja jednu vezu asocijacije koja postoji ka drugoj tabeli RDFTriple i sadrži sve trojke koje pripadaju RDF grafu identifikovanim uz pomoć RDFGraphId atributa.

RDF triple entitet kao primarni ključ u okviru tabele baze podataka sadrži RDFTripleId, koji kao i u slučaju RDF graf entiteta predstavlja jedinstveni broj koji se dodeljuje svakoj posebnoj RDF trojki koja

se skladišti u okviru MySQL ili PostgreSQL baze podataka.

RDFGraphId predstavlja strani ključ, koji identifikuje kojem RDF grafu pripada svaka od trojki.

Subject, Predicate, Object odnose se na subjekat, predikat i objekat svake od RDF trojki, dok GraphName se odnosi na ime RDF grafa kojem ta RDF trojka pripada.

Sledeći korak je kreiranje DbContext klase, koja predstavlja sesiju sa bazom podataka i preko koje je omogućeno postavljanje upita, razmena i skladištenje podataka, ona je takođe zadužena da prati promene izvršene nad objektima i održava konekciju sa bazom podataka.

Na slici 3.3.1 prikazana je mySqlDbContext klasa korišćena u okviru ove aplikacije, ona predstavlja konkretnu klasu koja nasleđuje DbContext klasu i koristeći se radi ostvarivanja konekcije sa mySQL bazom podataka.

```
5 references
public class mySqlDbContext : DbContext
{
    1 reference
    public mySqlDbContext() : base("mySqlDbContext")
    {
        ...
    }
    1 reference
    public DbSet<RDFGraph> RDFGraphs { get; set; }
    1 reference
    public DbSet<RDFTriple> RDFTriples { get; set; }
}
```

Slika 3.3.1 prikaz izgleda mySqlDbContext klase

DbSet<RDFGraph> i DbSet<RDFTriple> predstavljaju dva propertyja ove klase i oni odgovaraju tabelama u okviru određene baze podataka, odnosno oni govore EntityFramework-u da postoji mapiranje između ovih objekata i tabele u bazi, a polja RDFGraph i RDFTriple klasa odgovaraju redovima u tim tabelama i kao takvi RDFGraphs i RDFTriples propertyji mogu da se iskoriste za izvršavanje CRUD operacija.

Konačni korak je konfiguracija migracija kao način upravljanja izmenama nad šemom baze podataka, odnosno migracije opisuju kako bi šema baze podataka trebala biti modifikovana tako da odražava promene iz aplikacionog data modela.

Koraci konfiguracije migracija su niz skripti koje je potrebno izvršiti, a to su:

- Enable-Migrations
- Add-Migration [Ime migracije]
- Update-Database

Enable-Migration komanda omogućava upotrebu migracija nad aplikacionom DbContext klasom (npr. mySqlDbContext)

Add-Migration komanda se uglavnom izvršava kada postoji neka promena u okviru aplikacionog data modela i tada će EntityFramework da uporedi trenutni i prethodni data model i izgenerisati migracionu skriptu koja reprezentuje neophodne promene data modela kako bi se šema baza podataka mogla ažurirati.

Primer jedne verzije migracije koja će se primeniti nad mySQL bazom podataka prikazana je na slici 3.3.2.

U okviru Up() metode nalaze se komande koje će EntityFramework primeniti nad šemom baze podataka i koje će biti deo nove ažurirane šeme, kao što je na primer kreiranje dbo.RDFGraphs i dbo.RDFTriples tabele i gde je pored samog kreiranja ovih tabela prikazane i kreiranje polja, odnosno redova ovih tabela, njihov tip podataka i određene restrikcije vezane za to polje.

Na primer za RDFGraphId polje, ono je tipa integer (ceo broj), deo nullable:false naglašava da ovo polje ne sme biti nedefinisano i takođe ono je primarni ključ tabele dbo.RDFGraphs.

U okviru Down() metode nalaze se komande koje će se primeniti u okviru vraćanja šeme baze podataka u prethodno stanje, odnosno u ono stanje kojem je bila pre primene ove verzije migracije.

U primeru na slici 3.3.2 ovo podrazumeva brisanje stranih ključeva, indeksa kao i brisanje samih tabela.

```
2 references
public partial class mySQL : DbMigration
{
    0 references
    public override void Up()
    {
        CreateTable(
            "dbo.RDFGraphs",
            c => new
            {
                RDFGraphId = c.Int(nullable: false, identity: true),
                GraphName = c.String(unicode: false),
            })
            .PrimaryKey(t => t.RDFGraphId);

        CreateTable(
            "dbo.RDFTriples",
            c => new
            {
                RDFTripleId = c.Int(nullable: false, identity: true),
                RDFGraphId = c.Int(nullable: false),
                Subject = c.String(unicode: false),
                Predicate = c.String(unicode: false),
                Object = c.String(unicode: false),
                GraphName = c.String(unicode: false),
            })
            .PrimaryKey(t => t.RDFTripleId)
            .ForeignKey("dbo.RDFGraphs", t => t.RDFGraphId, cascadeDelete: true)
            .Index(t => t.RDFGraphId);
    }

    0 references
    public override void Down()
    {
        DropForeignKey("dbo.RDFTriples", "RDFGraphId", "dbo.RDFGraphs");
        DropIndex("dbo.RDFTriples", new[] { "RDFGraphId" });
        DropTable("dbo.RDFTriples");
        DropTable("dbo.RDFGraphs");
    }
}
```

Slika 3.3.2 primer jedne verzije migracione skripte koja će se primeniti nad mySQL bazom podataka Update-Database komanda izvršava migracionu skriptu koja je definisana nakon upotrebe Add-Migration komande i samim tim se ažurira navedena šema baze podataka.

Nakon ovih koraka, pošto je realizovana konekcija između aplikacije i relacionih baza podataka može da se pređe na testiranja performansi CRUD operacija i SQL upita ovih baza.

Testiranje performansi, uz pomoć BenchmarkDotNet biblioteke, kao i u slučaju Jena RDF skladišta može da se podeli na dva dela:

- Inicijalizaciju neophodnih objekata pre neposrednog izvršavanja merenja
- Merenje performansi CRUD operacija i SQL upita prilikom interakcije sa mySQL i PostgreSQL bazom podataka

#### 4.3.2.1 INICIJALIZACIJA OBJEKATA PRE NEPOSREDNOG IZVRŠAVANJA MERENJA

Inicijalizacija neophodnih objekata pre neposrednog izvršavanja merenja potpuno u slučaju relacionih baza podataka potpuno je analogna kao i u slučaju rada sa Jena RDF skladištem.

Na slici 3.3.3 vidi se Setup() funkcija, koja se koristi u okviru inicijalizacije objekata mySQL i PostgreSQL skladišta.



Inicijalizuju se sledeći objekti:

- dbContext predstavlja objekat DbContext klase, koja reprezentuje sesiju sa mySQL ili PostgreSQL bazom podataka i uz pomoć koje se mogu izvršavati CRUD ili SQL operacije, koristeći LINQ upite koje definiše EntityFramework I čija je sintaksa integrisana u okviru C# jezika
- handler predstavlja objekat, koji se koristi za parsiranje RDF fajlova, odnosno RDF trojki
- testGraphs predstavlja objekat koji sadrži RDF fajlove različitih veličina i broja RDF trojki koji će se koristiti za potrebe merenja performansi
- updatedTriples predstavlja objekat koji sadrži RDF trojke koje će se koristiti za potrebe testiranja performansi prilikom ažuriranja određenih RDF trojki

```
[GlobalSetup]
0 references
public void Setup()
{
    dbContext = new MySQLDbContext();
    handler = new RDFFileHandler();

    testGraphs = new Dictionary<string, IGraph>
    {
        {"small", handler.LoadRDF("D:\\Webojsa_Knezevic_Projekat_PR_104_2015\\TestFiles\\modelLabs_PowerTransformer_Example.xml") },
        {"medium", handler.LoadRDF("D:\\Webojsa_Knezevic_Projekat_PR_104_2015\\TestFiles\\CGMES_v2.4.15_RealGridTestConfiguration_SSH_V2.xml") },
        {"large", handler.LoadRDF("D:\\Webojsa_Knezevic_Projekat_PR_104_2015\\TestFiles\\CGMES_v2.4.15_RealGridTestConfiguration_EQ_V2.xml") }
    };

    updatedTriples = new List<RDFTriple> { new RDFTriple { RDFTripleId = 1, Subject = "UpdatedSubject1", Predicate = "UpdatedPredicate1", Object = "UpdatedObject1" },
                                           new RDFTriple { RDFTripleId = 2, Subject = "UpdatedSubject2", Predicate = "UpdatedPredicate2", Object = "UpdatedObject2" }
    };
}
```

Slika 3.3.3 Primer izgleda Global Setup funkcije za slučaj mySQL i PostgreSQL skladišta

#### 4.3.2.2 MERENJE PERFORMANSI CRUD OPERACIJA I SQL UPITA PRILIKOM INTERAKCIJE SA REALCIONIM BAZAMA PODATAKA

Merenje performansi CRUD operacija i SQL upita potpuno je analogno kao u slučaju merenja performansi Jena RDF skladišta.

Neophodno je samo definisati funkciju koja se testira i uz pomoć BenchmarkDotNet biblioteke, koristeći attribute [Benchmark] i [Arguments] obeležiti funkciju koju je potrebno testirati, odnosno definisati konstante koje će se mapirati na parametre testirane funkcije.

Na slici 3.3.4 je primer merenja performansi skladištenja RDF grafa sa svojim pridruženim RDF trojkama.

Testira se RDF graf sa malim, srednjim i velikim broj RDF trojki, ti grafovi i njihov base URI definisan je uz pomoć [Arguments] atributa.

Nakon toga kreira se novi RDFGraph objekat, definisan u okviru aplikacionog data modela, čija se polja popunjavaju podacima RDF fajla, koji želimo da skladištimo u okviru mySQL ili PostgreSQL baze podataka, pa se tako GraphName string polje popunjava rdfGraph.BaseUri poljem ukoliko ono nije null, odnosno "DefaultGraph" stringom ukoliko levi izraz jeste null.

DbContext objekat potom se koristi da se pristupi DbSet svojstvima koji predstavljaju entitete, dbContext takođe grupiše sve izmene u okviru ovog objekta koje će zatim biti primenjene na samu bazu podataka.

Na slici 3.3.4 u okviru RDFGraphs entite dodaje se graphDB objekat koji predstavlja novi RDFGraph objekat koji želimo da skladištimo u okviru baze podataka, dok uz pomoć SaveChanges() metode sve izmene koje su grupisane u okviru dbContext objekta se izvršavaju, što rezultuje dodavanjem objekta definisanim uz pomoć RDFGraph objekta u RDFGraphs tabelu u bazi podataka.

Nakon dodavanja samog RDF grafa, njemu se pridružuju i sve RDF trojke koje mu pripadaju, pa tako koristeći LINQ za svaku trojku koja pripada RDF fajlu kreira se novi RDFTriple objekat čija se polja popunjavaju na osnovu trojki iz RDF fajla i zatim dodavaju u RDFTriples tabelu mySQL ili PostgreSQL baze podataka.

```

[Benchmark]
[Arguments("http://iec.ch/TC57/2013/CIM-schema-cim15#", "modelLabs_PowerTransformer_Example.xml")]
[Arguments("http://iec.ch/TC57/2013/CIM-schema-cim16#", "CGMES_v2.4.15_RealGridTestConfiguration_SSH_V2")]
[Arguments("http://iec.ch/TC57/2013/CIM-schema-cim16#", "CGMES_v2.4.15_RealGridTestConfiguration_EQ_V2")]
0 references
public void AddRDFToDB(string baseUri, string graphName)
{
    IGraph rdfGraph = GetGraph(graphName);

    rdfGraph.BaseUri = new Uri(baseUri + graphName);

    var graphDB = new RDFGraph { GraphName = rdfGraph.BaseUri?.ToString() ?? "DefaultGraph" };

    dbContext.RDFGraphs.Add(graphDB);
    dbContext.SaveChanges();

    var triples = rdfGraph.Triples.Select(t => new RDFTriple
    {
        RDFGraphId = graphDB.RDFGraphId,
        Subject = t.Subject.ToString(),
        Predicate = t.Predicate.ToString(),
        Object = t.Object.ToString(),
        GraphName = t.GraphUri?.ToString() ?? graphDB.GraphName
    }).ToList();

    graphDB.Triples = triples;

    dbContext.RDFTriples.AddRange(triples);
    dbContext.SaveChanges();
}

```

Slika 3.3.4 Primer skladištenja RDF grafa i njegovih RDF trojki u okviru mySQL ili PostgreSQL baze podataka

Na slici 3.3.5 je primer merenja performansi čitanja svih RDF trojki za određeni RDF graf.

Kako bi se identifikovao traženi RDF graf, potrebno je kao ulazni parametar uneti Id tog grafa, a to je određeno u ovom primeru uz pomoć [Arguments] atributa.

Funkcija vraća kao povratnu vrednost ime traženog RDF grafa kao i listu svih RDF trojki koje mu pripadaju.

Koristeći dbContext i LINQ pristupa se RDFGraphs tabeli u bazi podataka i zatim traži prvi RDF graf sa RDFGraphId vrednošću koji je jednak vrednosti graphId objekata.

Nakon što je taj RDF graf pronađen kreira se novi RDFGraphs objekat i njegova polja se postavljaju na vrednosti polja traženog RDF grafa ako on postoji.

```

[Benchmark]
[Arguments(1)]
0 references
public (string graphName, List<RDFTriple> triples) ReadTriplesForGraph(int graphId)
{
    var result = dbContext.RDFGraphs
        .Where(g => g.RDFGraphId == graphId)
        .Select(g => new
        {
            g.GraphName,
            Triples = g.Triples
        })
        .FirstOrDefault();

    if (result != null)
    {
        return (result.GraphName, result.Triples.ToList());
    }

    return (null, null);
}

```

Slika 3.3.5 Primer čitanja svih RDF trojki za određeni RDF graf

Na slici 3.3.6 je primer merenja performansi ažuriranja RDF trojki na osnovu izabranog RDF grafa.

Analogno prethodnim funkcijama prvo se pronalazi RDF graf koji je identifikovan graphId objektom, nakon pronalaska tog RDF grafa graphDB objekat se postavlja na vrednost pronađenog RDF grafa, postavljanjem polja GraphName i Triples sa vrednostima tih polja traženog RDF grafa.

Ako takav RDF graf postoji foreach komandom prolazi se kroz sve RDF trojke tog grafa i pokušavaju da se nađu RDF trojke koje je neophodno ažurirati, a čiji RDFTripleId i subject, predicate, object polja su postavljena u okviru Setup() funkcije.

Ako takve RDF trojke postoje u bazi subject, predicate, object polja se ažuriraju vrednostima koja su navedene u okviru updatedTriple objekta i zatim uz pomoć metode SaveChanges() dbContext objekta mySQL ili PostgreSQL baza podataka se ažurira.

```
[Benchmark]
[Arguments(1)]
0 references
public void UpdateRDFInDB(int graphId)
{
    var graphDB = dbContext.RDFGraphs
        .Where(g => g.RDFGraphId == graphId)
        .Select(g => new
        {
            g.GraphName,
            Triples = g.Triples
        })
        .FirstOrDefault();

    if(graphDB != null)
    {
        foreach(var updatedTriple in updatedTriples)
        {
            var existingTriple = graphDB.Triples.FirstOrDefault(t => t.RDFTripleId == updatedTriple.RDFTripleId);

            if(existingTriple != null)
            {
                existingTriple.Subject = updatedTriple.Subject;
                existingTriple.Predicate = updatedTriple.Predicate;
                existingTriple.Object = updatedTriple.Object;
            }
        }

        dbContext.SaveChanges();
    }
}
```

Slika 3.3.6 Primer ažuriranja RDF trojki izabranog RDF grafa u okviru mySQL ili PostgreSQL baze podataka

Što se tiče operacije brisanja u slučaju relacionih baza podataka definisane su dva tipa brisanja:

- Brisanje RDF trojki iz određenog RDF grafa
- Brisanje RDF grafa i RDF trojki iz obe tabele

Na slici 3.3.7 je primer brisanja RDF trojki iz određenog RDF grafa.

```
[Benchmark]
[Arguments(1)]
0 references
public void DeleteTriplesFromGraph(int graphId)
{
    var triplesToRemove = dbContext.RDFTriples
        .Where(t => t.RDFGraphId == graphId)
        .ToList();

    dbContext.RDFTriples.RemoveRange(triplesToRemove);

    dbContext.SaveChanges();
}
```

Slika 3.3.7 Primer brisanja RDF trojki iz određenog traženog RDF grafa

Kao i do sada prvi deo koda pronalazi RDF graf iz kojeg je potrebno obrisati određene RDF trojke, a zatim iz RDFTriples tabele brišu se sve RDF trojke koje su deo traženog grafa.

Na slici 3.3.8 je primer brisanja svih RDF trojki iz određenog grafa, a potom brisanje i samom RDF grafa.

Kod je identičan prethodnom načinu brisanja s tim da je dodana još jedna linija koda koja iz tabele RDFGraphs briše i sam RDF graf čiji RDFGraphId je jednak vrednosti graphId objekta.

```
[Benchmark]
[Arguments(2)]
0 references
public void DeleteGraphAndTriples(int graphId)
{
    var graphDB = dbContext.RDFGraphs
        .FirstOrDefault(g => g.RDFGraphId == graphId);

    if (graphDB != null)
    {
        var triplesToRemove = dbContext.RDFTriples
            .Where(t => t.RDFGraphId == graphId)
            .ToList();

        dbContext.RDFTriples.RemoveRange(triplesToRemove);

        dbContext.RDFGraphs.Remove(graphDB);

        dbContext.SaveChanges();
    }
}
```

Slika 3.3.8 Primer brisanja RDF trojki iz određenog traženog RDF grafa, a potom i samog RDF grafa

#### 4.3.3 REZULTATI MERENJA PERFORMANSI JENA RDF, mySQL i PostgreSQL SKLADIŠTA I NJIHOVO UPOREĐIVANJE

BenchmarkDotNet biblioteka pored samog merenja performansi pruža i sumiranje rezultata u različitim formatima, ovi izlazni formati mogu da se podese korišćenjem Exporter atributima kao što je na primer [CSVExporter] koji sumira rezultat u .csv formatu.

Rezultati merenja uveliko zavise od stvari kao što su:

- Specifikacije računara na kojem se merenje vrši (CPU, RAM memorija, operativni sistem itd..)
- Verzija .NET frejmworka
- Uticaj pozadinskih programa

Zbog navedenih stvari ove informacije se navode pre samih rezultata merenja kako bi se ona mogla sagledati u takvom kontekstu.

BenchmarkDotNet sve izveštaje rezultate merenja čuva u okviru results foldera na putanji /bin/Release/BenchmarkDotNetArtifacts/results.

Testiranja da bi se dobila tačna merenja potrebno je izvršavati u Release modu bez debugera, jer je tada uključena optimizacija koda `<Optimize>true</Optimize>` u okviru csproj fajla i nema uticaja na performanse koje uključuje Debug mod.

U okviru ove aplikacije prilikom sumiranja rezulta izlazni formati koji se koriste su .html, .md, .csv.

Na slici 3.3.9, 3.4 i 3.4.1 su prikazani rezultati merenja performansi za CRUD i SPARQL operacije prilikom interakcije sa Jena RDF skladištem prilikom rada sa RDF fajlom male, srednje i velike veličine.

Iznad same tabele prikazane su informacije o verziji BenchmarkDotNet biblioteke, operativnom sistemu, CPU, kao i verziji .NET frejmworka.

Kolone sumiranih rezultata:

- **Method** - Imena metode čije se performanse mere
- **baseUri** - Uri na osnovu kojeg se rešavaju svih ostali relativni Uri-ji za određeni RDF graf
- **Mean** - Srednje vrednost brzine izvršavanja metode
- **Error** – Polovina od 99.9% intervala poverenja
- **StdDev** – Standardna devijacija od svih merenja
- **Median** – Vrednost koja razdvaja višu polovinu svih merenja
- **Gen0** – Garbage kolektor generacije 0 kolekcije po 1000 operacija
- **Gen1** - Garbage kolektor generacije 1 kolekcije po 1000 operacija
- **Gen2** - Garbage kolektor generacije 2 kolekcije po 1000 operacija
- **Allocated** – Količina alocirane memorije po jednoj operaciji
- **1  $\mu$  s** - 1 Microsecond (0.000001 sec)

BenchmarkDotNet v0.13.8, Windows 10 (10.0.19045.3693/22H2/2022Update)  
Intel Core i5-8400 CPU 2.803Hz (Coffee Lake), 1 CPU, 6 logical and 6 physical cores  
[Host] : .NET Framework 4.8.1 (4.8.9181.0), X86 LegacyJIT [AttachedDebugger]  
DefaultJob : .NET Framework 4.8.1 (4.8.9181.0), X86 LegacyJIT

| Method                              | baseUri              | graphName            | Mean                | Error             | StdDev            | Median              | Gen0     | Gen1     | Gen2    | Allocated  |
|-------------------------------------|----------------------|----------------------|---------------------|-------------------|-------------------|---------------------|----------|----------|---------|------------|
| DeleteGraph_InMemory                | http:(...)im15# [41] | model(...)e.xml [38] | 859.0 $\mu$ s       | 17.08 $\mu$ s     | 47.63 $\mu$ s     | 846.4 $\mu$ s       | -        | -        | -       | 8.89 KB    |
| DeleteGraph_TDB2                    | http:(...)im15# [41] | model(...)e.xml [38] | 787,394.6 $\mu$ s   | 13,018.30 $\mu$ s | 13,368.83 $\mu$ s | 787,568.6 $\mu$ s   | -        | -        | -       | 24 KB      |
| LoadGraph_InMemory                  | http:(...)im15# [41] | model(...)e.xml [38] | 13,449.2 $\mu$ s    | 445.35 $\mu$ s    | 1,219.14 $\mu$ s  | 13,101.7 $\mu$ s    | 296.8750 | 93.7500  | -       | 1419.42 KB |
| LoadGraph_TDB2                      | http:(...)im15# [41] | model(...)e.xml [38] | 12,607.7 $\mu$ s    | 251.25 $\mu$ s    | 360.33 $\mu$ s    | 12,590.3 $\mu$ s    | 296.8750 | 46.8750  | 15.6250 | 1419.51 KB |
| SaveGraph_InMemory                  | http:(...)im15# [41] | model(...)e.xml [38] | 9,866.8 $\mu$ s     | 238.08 $\mu$ s    | 694.48 $\mu$ s    | 9,813.4 $\mu$ s     | 200.0000 | 200.0000 | -       | 1003.49 KB |
| SaveGraph_TDB2                      | http:(...)im15# [41] | model(...)e.xml [38] | 1,394,450.1 $\mu$ s | 18,466.54 $\mu$ s | 17,273.61 $\mu$ s | 1,392,063.4 $\mu$ s | -        | -        | -       | 1012.13 KB |
| SparqlComplexQuerySmallRDF_InMemory | http:(...)im15# [41] | model(...)e.xml [38] | 18,435.9 $\mu$ s    | 365.24 $\mu$ s    | 639.68 $\mu$ s    | 18,301.1 $\mu$ s    | 125.0000 | 125.0000 | -       | 661.26 KB  |
| SparqlComplexQuerySmallRDF_TDB2     | http:(...)im15# [41] | model(...)e.xml [38] | 5,976.7 $\mu$ s     | 118.42 $\mu$ s    | 98.88 $\mu$ s     | 5,978.9 $\mu$ s     | 78.1250  | 78.1250  | -       | 367.83 KB  |
| SparqlSimpleQuerySmallRDF_InMemory  | http:(...)im15# [41] | model(...)e.xml [38] | 4,685.3 $\mu$ s     | 190.85 $\mu$ s    | 556.73 $\mu$ s    | 4,603.9 $\mu$ s     | 15.6250  | 15.6250  | -       | 73.78 KB   |
| SparqlSimpleQuerySmallRDF_TDB2      | http:(...)im15# [41] | model(...)e.xml [38] | 4,877.3 $\mu$ s     | 216.68 $\mu$ s    | 618.19 $\mu$ s    | 4,820.9 $\mu$ s     | 15.6250  | 15.6250  | -       | 73.78 KB   |
| UpdateGraphAddTriples_InMemory      | http:(...)im15# [41] | model(...)e.xml [38] | 6,850.2 $\mu$ s     | 136.36 $\mu$ s    | 182.04 $\mu$ s    | 6,784.8 $\mu$ s     | 101.5625 | 101.5625 | -       | 480.66 KB  |
| UpdateGraphAddTriples_TDB2          | http:(...)im15# [41] | model(...)e.xml [38] | 797,000.3 $\mu$ s   | 11,352.64 $\mu$ s | 10,063.82 $\mu$ s | 796,146.5 $\mu$ s   | -        | -        | -       | 491.26 KB  |
| UpdateGraphDeleteTriples_InMemory   | http:(...)im15# [41] | model(...)e.xml [38] | 6,730.0 $\mu$ s     | 132.30 $\mu$ s    | 228.21 $\mu$ s    | 6,717.7 $\mu$ s     | 109.3750 | 31.2500  | -       | 560.83 KB  |
| UpdateGraphDeleteTriples_TDB2       | http:(...)im15# [41] | model(...)e.xml [38] | 818,548.3 $\mu$ s   | 16,060.53 $\mu$ s | 18,495.33 $\mu$ s | 821,822.4 $\mu$ s   | -        | -        | -       | 491.26 KB  |

**Slika 3.3.9 Rezultati merenja CRUD i SPARQL operacija Jena RDF skladišta u slučaju RDF fajla male veličine**

BenchmarkDotNet v0.13.8, Windows 10 (10.0.19045.3693/22H2/2022Update)  
Intel Core i5-8400 CPU 2.803Hz (Coffee Lake), 1 CPU, 6 logical and 6 physical cores  
[Host] : .NET Framework 4.8.1 (4.8.9181.0), X86 LegacyJIT  
DefaultJob : .NET Framework 4.8.1 (4.8.9181.0), X86 LegacyJIT

| Method                               | baseUri              | graphName            | Mean                | Error              | StdDev             | Median              | Gen0       | Gen1       | Gen2      | Allocated    |
|--------------------------------------|----------------------|----------------------|---------------------|--------------------|--------------------|---------------------|------------|------------|-----------|--------------|
| DeleteGraph_InMemory                 | http:(...)im16# [41] | CGMES(...)SH_V2 [46] | 685.6 $\mu$ s       | 13.69 $\mu$ s      | 31.18 $\mu$ s      | 673.3 $\mu$ s       | -          | -          | -         | 8.97 KB      |
| DeleteGraph_TDB2                     | http:(...)im16# [41] | CGMES(...)SH_V2 [46] | 676,536.1 $\mu$ s   | 13,121.48 $\mu$ s  | 12,273.84 $\mu$ s  | 673,217.9 $\mu$ s   | -          | -          | -         | 24 KB        |
| LoadGraph_InMemory                   | http:(...)im16# [41] | CGMES(...)SH_V2 [46] | 3,371,274.7 $\mu$ s | 38,796.03 $\mu$ s  | 36,289.83 $\mu$ s  | 3,371,251.1 $\mu$ s | 73000.0000 | 49000.0000 | -         | 442263.38 KB |
| LoadGraph_TDB2                       | http:(...)im16# [41] | CGMES(...)SH_V2 [46] | 3,352,911.3 $\mu$ s | 41,692.06 $\mu$ s  | 38,998.78 $\mu$ s  | 3,346,130.0 $\mu$ s | 73000.0000 | 49000.0000 | -         | 442401.6 KB  |
| SaveGraph_InMemory                   | http:(...)im16# [41] | CGMES(...)SH_V2 [46] | 2,451,931.2 $\mu$ s | 9,988.02 $\mu$ s   | 8,340.45 $\mu$ s   | 2,450,178.6 $\mu$ s | 73000.0000 | 13000.0000 | 1000.0000 | 365893.45 KB |
| SaveGraph_TDB2                       | http:(...)im16# [41] | CGMES(...)SH_V2 [46] | 7,520,957.9 $\mu$ s | 112,473.30 $\mu$ s | 105,207.59 $\mu$ s | 7,553,708.7 $\mu$ s | 73000.0000 | 20000.0000 | -         | 365885.81 KB |
| SparqlComplexQueryMediumRDF_InMemory | http:(...)im16# [41] | CGMES(...)SH_V2 [46] | 56,755.4 $\mu$ s    | 640.77 $\mu$ s     | 568.03 $\mu$ s     | 56,630.0 $\mu$ s    | 555.5556   | 222.2222   | -         | 3399.03 KB   |
| SparqlComplexQueryMediumRDF_TDB2     | http:(...)im16# [41] | CGMES(...)SH_V2 [46] | 169,640.9 $\mu$ s   | 1,052.88 $\mu$ s   | 822.02 $\mu$ s     | 169,611.5 $\mu$ s   | 666.6667   | 333.3333   | -         | 3402.85 KB   |
| SparqlSimpleQueryMediumRDF_InMemory  | http:(...)im16# [41] | CGMES(...)SH_V2 [46] | 167,801.6 $\mu$ s   | 903.79 $\mu$ s     | 801.19 $\mu$ s     | 167,932.4 $\mu$ s   | 1500.0000  | 1500.0000  | -         | 9502.61 KB   |
| SparqlSimpleQueryMediumRDF_TDB2      | http:(...)im16# [41] | CGMES(...)SH_V2 [46] | 316,152.3 $\mu$ s   | 3,151.41 $\mu$ s   | 2,947.83 $\mu$ s   | 315,541.2 $\mu$ s   | 1000.0000  | -          | -         | 9640.36 KB   |
| UpdateGraphAddTriples_InMemory       | http:(...)im16# [41] | CGMES(...)SH_V2 [46] | 7,164.5 $\mu$ s     | 136.16 $\mu$ s     | 234.87 $\mu$ s     | 7,170.2 $\mu$ s     | 93.7500    | -          | -         | 482.74 KB    |
| UpdateGraphAddTriples_TDB2           | http:(...)im16# [41] | CGMES(...)SH_V2 [46] | 677,734.5 $\mu$ s   | 13,212.08 $\mu$ s  | 15,728.04 $\mu$ s  | 677,456.4 $\mu$ s   | -          | -          | -         | 492.27 KB    |
| UpdateGraphDeleteTriples_InMemory    | http:(...)im16# [41] | CGMES(...)SH_V2 [46] | 7,301.3 $\mu$ s     | 145.17 $\mu$ s     | 142.57 $\mu$ s     | 7,260.1 $\mu$ s     | 101.5625   | 7.8125     | -         | 482.67 KB    |
| UpdateGraphDeleteTriples_TDB2        | http:(...)im16# [41] | CGMES(...)SH_V2 [46] | 724,210.6 $\mu$ s   | 26,285.90 $\mu$ s  | 73,274.59 $\mu$ s  | 700,116.1 $\mu$ s   | -          | -          | -         | 489.93 KB    |

**Slika 3.4 Rezultati merenja CRUD i SPARQL operacija Jena RDF skladišta u slučaju RDF fajla srednje veličine**

BenchmarkDotNet v0.13.8, Windows 10 (10.0.19045.3803/22H2/2022Update)  
 Intel Core i5-8400 CPU 2.80GHz (Coffee Lake), 1 CPU, 6 logical and 6 physical cores  
 [Host] : .NET Framework 4.8.1 (4.8.9181.0), X86 LegacyJIT  
 DefaultJob : .NET Framework 4.8.1 (4.8.9181.0), X86 LegacyJIT

| Method                              | baseUri              | graphName            | Mean            | Error         | StdDev        | Median          | Gen0        | Gen1        | Gen2     | Allocated     |
|-------------------------------------|----------------------|----------------------|-----------------|---------------|---------------|-----------------|-------------|-------------|----------|---------------|
| DeleteGraph_InMemory                | http://...im16# [41] | CGMES(...)EQ_V2 [45] | 690.8 µs        | 8.61 µs       | 14.38 µs      | 687.9 µs        | 5.8594      | 3.9063      | -        | 41.28 KB      |
| DeleteGraph_TDB2                    | http://...im16# [41] | CGMES(...)EQ_V2 [45] | 787,796.7 µs    | 12,560.61 µs  | 10,488.68 µs  | 787,761.8 µs    | -           | -           | -        | 24 KB         |
| LoadGraph_InMemory                  | http://...im16# [41] | CGMES(...)EQ_V2 [45] | NA              | NA            | NA            | NA              | NA          | NA          | NA       | NA            |
| LoadGraph_TDB2                      | http://...im16# [41] | CGMES(...)EQ_V2 [45] | NA              | NA            | NA            | NA              | NA          | NA          | NA       | NA            |
| SaveGraph_InMemory                  | http://...im16# [41] | CGMES(...)EQ_V2 [45] | 24,522,186.6 µs | 216,738.25 µs | 169,214.97 µs | 24,481,931.1 µs | 833000.0000 | 141000.0000 | -        | 4194650.41 KB |
| SaveGraph_TDB2                      | http://...im16# [41] | CGMES(...)EQ_V2 [45] | 60,212,533.9 µs | 288,141.64 µs | 269,527.87 µs | 60,170,346.7 µs | 840000.0000 | 748000.0000 | -        | 4194666.08 KB |
| SpqrqlComplexQueryLargeRDF_InMemory | http://...im16# [41] | CGMES(...)EQ_V2 [45] | 11,231.6 µs     | 206.36 µs     | 161.11 µs     | 11,214.7 µs     | -           | -           | -        | 114.46 KB     |
| SpqrqlComplexQueryLargeRDF_TDB2     | http://...im16# [41] | CGMES(...)EQ_V2 [45] | 88,774.2 µs     | 444.86 µs     | 394.35 µs     | 88,734.2 µs     | -           | -           | -        | 116 KB        |
| SpqrqlSimpleQueryLargeRDF_InMemory  | http://...im16# [41] | CGMES(...)EQ_V2 [45] | 787,015.7 µs    | 93,971.45 µs  | 258,824.67 µs | 889,066.0 µs    | 1500.0000   | 1000.0000   | 500.0000 | 9812.06 KB    |
| SpqrqlSimpleQueryLargeRDF_TDB2      | http://...im16# [41] | CGMES(...)EQ_V2 [45] | 779,793.9 µs    | 15,417.63 µs  | 17,754.97 µs  | 788,145.7 µs    | 1000.0000   | 1000.0000   | -        | 9822.71 KB    |
| UpdateGraphAddTriples_InMemory      | http://...im16# [41] | CGMES(...)EQ_V2 [45] | 7,592.3 µs      | 141.22 µs     | 132.10 µs     | 7,546.5 µs      | 93.7500     | 15.6250     | -        | 520.89 KB     |
| UpdateGraphAddTriples_TDB2          | http://...im16# [41] | CGMES(...)EQ_V2 [45] | 793,102.1 µs    | 9,265.72 µs   | 7,737.30 µs   | 792,021.0 µs    | -           | -           | -        | 490.78 KB     |
| UpdateGraphDeleteTriples_InMemory   | http://...im16# [41] | CGMES(...)EQ_V2 [45] | 6,823.0 µs      | 91.35 µs      | 85.45 µs      | 6,869.8 µs      | 101.5625    | 23.4375     | -        | 527.41 KB     |
| UpdateGraphDeleteTriples_TDB2       | http://...im16# [41] | CGMES(...)EQ_V2 [45] | 791,616.1 µs    | 8,119.53 µs   | 6,780.18 µs   | 793,058.3 µs    | -           | -           | -        | 492.15 KB     |

Slika 3.4.1 Rezultati merenja CRUD i SPARQL operacija Jena RDF skladišta u slučaju RDF fajla velike veličine

Na slici 3.4.2 i 3.4.3 su prikazani rezultati merenja performansi za CRUD i SQL operacije prilikom interakcije sa mySQL skladištem prilikom rada sa RDF fajlom male i srednje veličine. U slučaju mySQL skladišta nije uključeno testiranje performansi CRUD i SQL operacija u radu sa RDF fajlom velike veličine, jer je prilikom testiranja došlo do preopterećenja memorije.

BenchmarkDotNet v0.13.8, Windows 10 (10.0.19045.3693/22H2/2022Update)  
 Intel Core i5-8400 CPU 2.80GHz (Coffee Lake), 1 CPU, 6 logical and 6 physical cores  
 [Host] : .NET Framework 4.8.1 (4.8.9181.0), X86 LegacyJIT  
 DefaultJob : .NET Framework 4.8.1 (4.8.9181.0), X86 LegacyJIT  
 Job-FTOBUC : .NET Framework 4.8.1 (4.8.9181.0), X86 LegacyJIT

| Method                 | Mean         | Error       | StdDev      | Gen0      | Gen1      | Allocated   |
|------------------------|--------------|-------------|-------------|-----------|-----------|-------------|
| SQLQueryMoreResults    | 5,975.6 µs   | 15.92 µs    | 14.89 µs    | 273.4375  | 273.4375  | 1292.89 KB  |
| ReadTriplesForGraph    | 6,587.9 µs   | 47.71 µs    | 42.30 µs    | 218.7500  | 218.7500  | 1009.01 KB  |
| UpdateRDFInDB          | 7,133.7 µs   | 17.86 µs    | 14.92 µs    | 226.5625  | 226.5625  | 1067.14 KB  |
| DeleteTriplesFromGraph | 856.5 µs     | 7.05 µs     | 5.88 µs     | 9.7656    | 9.7656    | 47.49 KB    |
| DeleteGraphAndTriples  | 721.6 µs     | 8.56 µs     | 7.15 µs     | 8.7891    | 8.7891    | 42.14 KB    |
| AddRDFToDB             | 212,669.8 µs | 4,222.61 µs | 8,433.01 µs | 2000.0000 | 1000.0000 | 10264.13 KB |
| SQLQueryFewResults     | 1,010.4 µs   | 9.74 µs     | 16.27 µs    | 17.5781   | 17.5781   | 86.85 KB    |

Slika 3.4.2 Rezultati merenja CRUD i SQL operacija mySQL skladišta u slučaju RDF fajla male veličine

BenchmarkDotNet v0.13.8, Windows 10 (10.0.19045.3693/22H2/2022Update)  
 Intel Core i5-8400 CPU 2.80GHz (Coffee Lake), 1 CPU, 6 logical and 6 physical cores  
 [Host] : .NET Framework 4.8.1 (4.8.9181.0), X86 LegacyJIT  
 DefaultJob : .NET Framework 4.8.1 (4.8.9181.0), X86 LegacyJIT  
 Job-FTOBUC : .NET Framework 4.8.1 (4.8.9181.0), X86 LegacyJIT

| Method                 | Mean                  | Error                | StdDev               | Gen0        | Gen1        | Gen2      | Allocated     |
|------------------------|-----------------------|----------------------|----------------------|-------------|-------------|-----------|---------------|
| SQLQueryMoreResults    | 1,640,213.3 $\mu$ s   | 8,720.11 $\mu$ s     | 7,730.15 $\mu$ s     | 83000.0000  | 83000.0000  | -         | 388405.75 KB  |
| ReadTriplesForGraph    | 1,713,263.0 $\mu$ s   | 12,747.82 $\mu$ s    | 9,952.66 $\mu$ s     | 64000.0000  | 1000.0000   | -         | 297822.57 KB  |
| UpdateRDFInDB          | 1,876,267.4 $\mu$ s   | 8,156.49 $\mu$ s     | 7,230.52 $\mu$ s     | 67000.0000  | 67000.0000  | -         | 315910.81 KB  |
| DeleteTriplesFromGraph | 1,332.3 $\mu$ s       | 26.18 $\mu$ s        | 50.44 $\mu$ s        | -           | -           | -         | 56 KB         |
| DeleteGraphAndTriples  | 1,189.3 $\mu$ s       | 20.57 $\mu$ s        | 18.24 $\mu$ s        | -           | -           | -         | 48 KB         |
| AddRDFToDB             | 264,280,990.4 $\mu$ s | 2,938,751.82 $\mu$ s | 2,748,910.28 $\mu$ s | 638000.0000 | 523000.0000 | 1000.0000 | 3258033.55 KB |
| SQLQueryFewResults     | 996.0 $\mu$ s         | 5.53 $\mu$ s         | 4.62 $\mu$ s         | 17.5781     | 17.5781     | -         | 86.92 KB      |

Slika 3.4.3 Rezultati merenja CRUD i SQL operacija mySQL skladišta u slučaju RDF fajla srednje veličine

Na slici 3.4.5 i 3.4.6 su prikazani rezultati merenja performansi za CRUD i SQL operacije prilikom interakcije sa PostgreSQL skladištem prilikom rada sa RDF fajlom male i srednje veličine. U slučaju PostgreSQL skladišta nije uključeno testiranje performansi CRUD i SQL operacija u radu sa RDF fajlom velike veličine, jer je prilikom testiranja došlo do preopterećenja memorije.

BenchmarkDotNet v0.13.8, Windows 10 (10.0.19045.3803/22H2/2022Update)  
 Intel Core i5-8400 CPU 2.80GHz (Coffee Lake), 1 CPU, 6 logical and 6 physical cores  
 [Host] : .NET Framework 4.8.1 (4.8.9181.0), X86 LegacyJIT  
 DefaultJob : .NET Framework 4.8.1 (4.8.9181.0), X86 LegacyJIT  
 Job-FTOBUC : .NET Framework 4.8.1 (4.8.9181.0), X86 LegacyJIT

| Method                 | Mean              | Error            | StdDev           | Median            | Gen0      | Gen1    | Allocated  |
|------------------------|-------------------|------------------|------------------|-------------------|-----------|---------|------------|
| SQLQueryNoJoin         | 1,285.6 $\mu$ s   | 25.45 $\mu$ s    | 49.63 $\mu$ s    | 1,270.7 $\mu$ s   | 19.5313   | 19.5313 | 95.02 KB   |
| ReadTriplesForGraph    | 1,590.8 $\mu$ s   | 31.55 $\mu$ s    | 60.79 $\mu$ s    | 1,590.0 $\mu$ s   | 17.5781   | 17.5781 | 89.52 KB   |
| UpdateRDFInDB          | 1,786.2 $\mu$ s   | 37.24 $\mu$ s    | 107.45 $\mu$ s   | 1,742.4 $\mu$ s   | 31.2500   | -       | 146.71 KB  |
| DeleteTriplesFromGraph | 414.1 $\mu$ s     | 7.83 $\mu$ s     | 14.32 $\mu$ s    | 411.9 $\mu$ s     | 6.8359    | -       | 32.54 KB   |
| DeleteGraphAndTriples  | 420.4 $\mu$ s     | 5.68 $\mu$ s     | 4.44 $\mu$ s     | 418.9 $\mu$ s     | 6.8359    | -       | 32.61 KB   |
| AddRDFToDB             | 116,571.0 $\mu$ s | 2,327.61 $\mu$ s | 4,960.33 $\mu$ s | 115,040.0 $\mu$ s | 1000.0000 | -       | 7048.05 KB |
| SQLQueryWithJoin       | 700.3 $\mu$ s     | 13.99 $\mu$ s    | 36.37 $\mu$ s    | 698.0 $\mu$ s     | 13.6719   | 13.6719 | 65.13 KB   |

Slika 3.4.5 Rezultati merenja CRUD i SQL operacija PostgreSQL skladišta u slučaju RDF fajla male veličine

BenchmarkDotNet v0.13.8, Windows 10 (10.0.19045.3803/22H2/2022Update)  
 Intel Core i5-8400 CPU 2.80GHz (Coffee Lake), 1 CPU, 6 logical and 6 physical cores  
 [Host] : .NET Framework 4.8.1 (4.8.9181.0), X86 LegacyJIT  
 Job-FTOBUC : .NET Framework 4.8.1 (4.8.9181.0), X86 LegacyJIT  
 DefaultJob : .NET Framework 4.8.1 (4.8.9181.0), X86 LegacyJIT

| Method                 | Mean                  | Error                | StdDev               | Median                | Gen0        | Gen1        | Gen2      | Allocated     |
|------------------------|-----------------------|----------------------|----------------------|-----------------------|-------------|-------------|-----------|---------------|
| AddRDFToDB             | 227,428,974.3 $\mu$ s | 2,912,161.72 $\mu$ s | 2,724,037.88 $\mu$ s | 228,549,393.7 $\mu$ s | 417000.0000 | 349000.0000 | 1000.0000 | 2241151.11 KB |
| DeleteGraphAndTriples  | 830.7 $\mu$ s         | 14.62 $\mu$ s        | 36.13 $\mu$ s        | 816.0 $\mu$ s         | -           | -           | -         | 40 KB         |
| DeleteTriplesFromGraph | 866.3 $\mu$ s         | 17.30 $\mu$ s        | 37.98 $\mu$ s        | 856.4 $\mu$ s         | -           | -           | -         | 40 KB         |
| ReadTriplesForGraph    | 347,381.4 $\mu$ s     | 6,902.26 $\mu$ s     | 11,145.86 $\mu$ s    | 343,183.2 $\mu$ s     | 2000.0000   | 2000.0000   | -         | 11793.02 KB   |
| SQLQueryNoJoin         | 266,710.7 $\mu$ s     | 5,058.27 $\mu$ s     | 4,731.51 $\mu$ s     | 268,608.8 $\mu$ s     | 3000.0000   | -           | -         | 17920.17 KB   |
| SQLQueryWithJoin       | 649.2 $\mu$ s         | 6.24 $\mu$ s         | 5.21 $\mu$ s         | 649.8 $\mu$ s         | 13.6719     | 13.6719     | -         | 65.13 KB      |
| UpdateRDFInDB          | 645,040.8 $\mu$ s     | 44,146.55 $\mu$ s    | 130,167.10 $\mu$ s   | 575,573.3 $\mu$ s     | 5000.0000   | 5000.0000   | -         | 29897.95 KB   |

Slika 3.4.6 Rezultati merenja CRUD i SQL operacija PostgreSQL skladišta u slučaju RDF fajla srednje veličine

Za testiranje performansi su u slučaju sva 3 skladišta, Jena RDF, mySQL i PostgreSQL, korišćeni isti RDF fajlovi male i srednje veličine:

- Male veličine (24KB) - modelLabs\_PowerTransformer\_Example.xml
- Srednje veličine (7.9MB) - CGMES\_v2.4.15\_RealGridTestConfiguration\_SSH\_V2.xml

U slučaju Jena RDF skladišta testiran je i RDF fajl velike veličine CGMES\_v2.4.15\_RealGridTestConfiguration\_EQ\_V2.xml (67 MB).

CRUD i SPARQL/SQL operacije su testirane pozivanjem posredno iz C# .NET konzolne aplikacije koristeći Entity Framework 6, pa samim tim rezultate treba interpretirati u okviru takvog konteksta.

U slučaju Jena RDF skladišta:

- Skladištenje RDF grafa (**operacija SaveGraph**) male veličine –
  - Srednje vreme brzine izvršavanja - efikasnije in-memory rešenje 140 puta
  - Memorijska efikasnost – minimalno efikasnije tdb2 rešenja (ne uvodi troškove garbage collector)
- Skladištenje RDF grafa (**operacija SaveGraph**) srednje veličine –
  - Srednje vreme brzine izvršavanja - efikasnije in-memory rešenje (pad sa 140 na samo 3 puta)
  - Memorijska efikasnost – nema značajnih razlika
- Čitanje RDF grafa iz skladišta (**operacija LoadGraph**) male i srednje veličine –
  - Srednje vreme brzine izvršavanja – minimalno efikasnije tdb2 rešenje
  - Memorijska efikasnost – minimalno efikasnije tdb2 rešenje
- Ažuriranje RDF grafa (**operacija UpdateGraph**) male i srednje veličine –
  - Srednje vreme brzine izvršavanja – efikasnije in-memory rešenje 100 puta
  - Memorijska efikasnost – nema značajnijih razlika
- Brisanje RDF grafa (**operacija DeleteGraph**) male i srednje veličine –
  - Srednje vreme brzine izvršavanja – efikasnije in-memory rešenja 900 puta
  - Memorijska efikasnost – efikasnije in-memory rešenje skoro 3 puta
- SPARQL upiti pri radu sa **RDF fajlom male veličine** –
  - Srednje vreme brzine izvršavanja – efikasnije tdb2 rešenje ili nema značajnih razlika u zavisnosti od samog SPARQL upita
  - Memorijska efikasnost – efikasnije tdb2 rešenje ili nema značajnih razlika u zavisnosti od samog SPARQL upita
- SPARQL upiti pri radu sa **RDF fajlom srednje veličine** –
  - Srednje vreme brzine izvršavanja – efikasnije in-memory rešenje 2 puta
  - Memorijska efikasnost – nema značajnih razlika

U slučaju RDF grafa velike veličine ne postoje značajne razlike u srednjoj brzini izvršavanja u slučaju ažuriranja RDF grafa (UpdateGraph) i brisanju RDF grafa (DeleteGraph) kada se on poredi sa RDF grafom srednje veličine.

Takođe može se ponovo primetiti da sa porastom veličine RDF fajla, odnosno broja RDF trojki se smanjuje relativna razlika između srednje brzine izvršavanja u slučaju tdb2 i in-memory rešenja i da kada se uzme u obzir da se u praksi radi sa milijardama RDF trojki, tdb2 rešenje postaje efikasnije kako se približavamo tom broju RDF trojki.

Što se tiče memorijske efikasnosti ne postoje značajne razlike između tdb2 i in-memory rešenja osim u slučaju brisanja RDF grafa, koje pokazuje da je tdb2 rešenje memorijski efikasnije.

Čitanje RDF grafa iz Jena RDF skladišta nije moguće, jer računar nema dovoljnu količinu memorije kako bi podržao testiranje takve operacije pri radu sa fajlom ove veličine.

U slučaju MySQL i PostgreSQL skladišta:

- Skladištenje RDF fajla (**operacija AddRDFToDB**) male i srednje veličine –
  - Srednje vreme brzine izvršavanja – efikasnije PostgreSQL skladište oko 1.5-2 puta
  - Memorijska efikasnosti – efikasnije PostgreSQL skladište 30-50%
- Čitanje RDF trojki za određeni RDF graf (**operacija ReadTriplesForGraph**) male veličine –
  - Srednje vreme brzine izvršavanja – efikasnije PostgreSQL skladište oko 4 puta



- Memorijska efikasnost - efikasnije PostgreSQL skladište 11 puta
- Čitanje RDF trojki za određeni RDF graf (**operacija ReadTriplesForGraph**) **srednje veličine** –
  - Srednje vreme brzine izvršavanja – efikasnije PostgreSQL skladište oko 5 puta
  - Memorijska efikasnost – efikasnije PostgreSQL skladište oko 25 puta
- Ažuriranje RDF trojki za određeni RDF graf (**operacija UpdateRDFInDB**) **male veličine** –
  - Srednje vreme brzine izvršavanja – efikasnije PostgreSQL skladište oko 4 puta
  - Memorijska efikasnost – efikasnije PostgreSQL skladište oko 7 puta
- Ažuriranje RDF trojki za određeni RDF graf (**operacija UpdateRDFInDB**) **srednje veličine** –
  - Srednje vreme brzine izvršavanja – efikasnije PostgreSQL skladište oko 3 puta
  - Memorijska efikasnost – efikasnije PostgreSQL skladište oko 10 puta
- Brisanje RDF trojki za određeni RDF graf (**operacija DeleteTriplesFromGraph**) **male i srednje veličine** –
  - Srednje vreme brzine izvršavanja – efikasnije PostgreSQL skladište 1.5-2 puta
  - Memorijska efikasnost – efikasnije PostgreSQL skladište oko 1.5 puta
- Brisanje RDF trojki i RDF grafa (**operacija DeleteGraphAndTriples**) **male i srednje veličine** –
  - Srednje vreme brzine izvršavanja – efikasnije PostgreSQL skladište oko 1.4-1.7 puta
  - Memorijska efikasnost - efikasnije PostgreSQL skladište oko 1.4 puta

U slučaju poređenja između Jena RDF skladišta i MySQL/PostgreSQL relacionih baza podataka u slučaju RDF fajlova malih i srednjih veličina vidi se da je Jena RDF skladište znatno efikasnije prilikom skladištenja RDF fajla i ažuriranja RDF trojki, pogotovo kako RDF fajl i broj RDF trojki raste.

Operacija brisanja RDF grafa i njegovih RDF trojki je približno efikasna između Jena RDF skladišta i PostgreSQL skladišta a oba su efikasnije od mySQL skladišta.

Operacija čitanja RDF grafa iz skladišta je efikasnija u slučaju MySQL/PostgreSQL skladišta u odnosu na Jena RDF skladište.

Operacija postavljanja SPARQL/SQL upita i njihova efikasnost zavisi od broja RDF trojki prisutnih u skladištu, s tim da što ima više prisutnih RDF trojki potrebno je više vremena da bi se upit izvršio pogotovo ako je potrebno vratiti veliki broj rezultata i taj odnos je jednak i za Jena RDF i za mySQL/PostgreSQL skladišta.

Samim tim zaključak je da je Jena RDF skladište znatno pogodnije za skladištenje RDF grafova/RDF trojki, pogotovo u slučaju velikih RDF fajlova. U slučaju manjih RDF fajlova treba koristiti in-memory rešenje, dok u slučaju velikih RDF fajlova tdb2 skladište je znatno efikasnije.

## 5 PREDLOZI ZA DALJA USAVRŠAVANJA

Prednosti aplikacije se ogledaju u tome što se korisniku uz pomoć jednostavnog korisničkog interfejsa pruža čitav spektar dostupnih akcija na jednom mestu. Od učitavanja CIM profila i CIM primera podataka do pružanja niza operacija manipulacija nad učitanih podataka bez potrebe njihovog ručnog modifikovanja, kao što je na primer omogućavanje izmena trojki izabranog RDF grafa direktno u aplikaciji.

Svaka akcija korisnika praćena je određenim povratnim odgovorom o uspešnosti ili neuspešnosti izvršavanja tražene akcije, kao i ponuđenim odgovorom kako taj nedostatak prevazići.

Performanse dostupnih operacija su izuzetno dobre za relativno male dokumente i male SPARQL upite i dovode do toga da aplikacija predstavlja dobro rešenje za proveru ispravnosti funkcionisanja eksperimentalnih distributivnih mreža.

Ipak glavna prednost aplikacije predstavlja uključivanje dotNetRDF reasoner-a i Jena RDFS reasoner-a, koji primenjuju bogat skup semantičkih pravila i zaključaka na osnovu unetog CIM profila i CIM primera podataka i tako daju ovim RDF trojkama dodatne informacije koje mogu biti korisne u cilju praćenja stanje distributivne mreže.

Dobra strana aplikacije je i ta što je aplikacija uvek ažurirana i reflektuje stanje podataka onakvo kakvo je u okviru Fuseki RDF skladišta, pa tako korisnik ima uvek uvid u podatke koji predstavljaju trenutno stanje distributivne mreže.

Osnovna mana je svakako to što performanse rešenja opadaju pri pokušaju učitavanja, parsiranja i skladištenja ogromnih RDF dokumenta. Daljim usavršavanjem projekta moguće je zaobići ovaj nedostatak. Umesto skladištenja velikog broja stringova u memoriji koje može biti sporo i neefikasno, drugačijom reprezentacijom ili konvertovanjem tipa podataka moguće je povećati performanse.

Takođe u slučaju konfiguracije Fuseki server, umesto memorijskog modela koji je nižih performansi i služi kao skladište za testiranje malih dokumenata bez mogućnosti perzistentnosti ovih podataka pri gašenju Fuseki servera moguće je koristiti tdb bazu podataka od strane Apache Jena. Ovo skladište je izuzetno visokih performansi i u slučaju skladištenja ogromnih RDF dokumenata, koristeći razne optimalne tehnike kao što su keširanje, indeksovanje podataka, korišćenje konkurentnosti itd. i omogućava perzistiranje podataka i prilikom gašenja Fuseki servera. Manja mana je i ta što nije moguća izmena Fuseki datasetova u runtime aplikacije, drugim rečima neophodno je konfigurisati config.ttl fajl pre samog startovanja Fuseki RDF skladišta gde prilikom svake sledeća izmena podrazumeva se da je neophodno ugasiti Fuseki server napraviti željenu izmenu i zatim ponovo pokrenuti Fuseki server kako bi ova izmena bila sačuvana.

Ujedno poslednja mana trebala bi da predstavlja i pravac daljeg istraživanja. Trenutno za .NET okruženje kada se koristi dotNetRDF biblioteka u sprezi sa Fuseki RDF skladištem nije moguće praviti pomenute izmene u runtime-u, neophodno je proširiti postojeći API metodama koje će omogućiti primenu ovih izmena kao što je to u slučaju za Jena Java okruženje.

Pravac daljeg istraživanja može biti i rešavanje problema gde prilikom skladištenja izuzetno velikih RDF dokumenata i pokušaja primene kompletnog skupa Jena RDFS Reasoner pravila radi dobijanja dodatnih zaključaka dolazi do velike degradacije performansi nakon što taj Reasoner pokuša da primeni pravila u odnosu na svaku trojku ontologije ili primera podataka.

Trenutno rešenje se ogleda u tome da se koristi pojednostavljeni RDFS Reasoner sa manjim obimom pravila koja mogu da se smatraju nepotrebnim u početku.

Nedostatak ovog pristupa predstavlja to što određene zaključke neće biti moguće izvršiti, pa samim tim i određene semantičke informacije će biti izostavljene.

Prednosti načina testiranja performansi opisanih u prethodnom delu je mogućnost pribavljanja informacija, koje omogućuju odabir pogodnog skladišta u zavisnosti od veličine RDF fajla, odnosno broja RDF trojki kako bi se izabralo najefikasnije rešenje, kao i u slučaju Jena RDF skladišta odabir in-memory ili tdb2 rešenja.

Glavna mana predstavljenog načina testiranja performansi je tačnost samih podataka, zbog raznih varijabli koje mogu da utiču na vrednost merenja, kao što su: operativni sistem, procesorska moć i

brzina i raspoloživost memorije računara, verzija .NET frejmvorka, broj i uticaj na procesorsko vreme pozadinskih procesa, pa se rezultati moraju interpretirati na osnovu ovog konteksta.

Takođe testiranje CRUD i SPARQL/SQL operacija korišćeno je posredstvom sintakse i operacija C# .NET Entity Frameworka i ona mogu dati drugačije rezultate u odnosu na specijalne alate i skripte Jena RDF i mySQL/PostgreSQL skladišta koje su specijalno stvorene kako bi ubrzale vreme neophodno za izvršavanje ovih operacija kao što su bulk loaderi, koji znatno mogu da poboljšaju performanse prilikom rada sa izuzetno velikim RDF fajlovima koji prevazilaze broj od milion RDF trojki ili više.

- [1] <https://cmte.ieee.org/pes-testfeeders/resources/>
- [2] **SERBIAN JOURNAL OF ELECTRICAL ENGINEERING** Vol. 15, No. 1, February 2018, 115-129 115 A  
**Brief Overview of the Distribution Test Grids with a Distributed Generation Inclusion Case Study**
- [3] **Y. Tina Lee, Information Modeling: From Design to Implementation, NIST, 1999**
- [4] **Common Information Model Primer: Third Edition. EPRI, Palo Alto, CA: 2015. 3002006001.**
- [5] **The Common Information Model for Distribution: An Introduction to the CIM for Integrating Distribution Applications and Systems. EPRI, Palo Alto, CA: 2008. 1016058.**
- [6] **MultiSpeak ® Version 3.0 User's Guide 1/1/2006, Gary A. McNaughton, P.E. Warren P. McNaughton, P.E. , Cornice Engineering, Inc. P.O. Box 2350 Pagosa Springs, Colorado 81147-2350**
- [7] <https://jena.apache.org/>
- [8] **An Introduction to IEC 61970-301 & 61968-11: The Common Information Model Institute for Energy and Environment Department of Electronic and Electrical Engineering University of Strathclyde Glasgow, UK January 2007**
- [9] **Common Information Model Primer: Third Edition. EPRI, Palo Alto, CA: 2015.**  
**3002006001 Section 5 IEC 62325-301 5-36**
- [10] **Common Information Model Primer: Third Edition. EPRI, Palo Alto, CA: 2015.**  
**3002006001 Section 5: CIM UML Translating a Circuit into CIM 5-10**
- [11] <https://benchmarkdotnet.org/>