ZADATAK DIPLOMSKOG – MASTER RADA

Zadatak rada je implementirati softverski alat za održavanje *CIM* modela definisanog u *Enterprise Architect*-u. Za realizaciju koristiti *API Enterprise Architect-a* i standardno softversko okruženje *Microsoft Visual Studio* i programski jezik *C#* kao i biblioteke koje nudi *Microsoft Office*. Omogućiti praćenje različitih verzija *CIM* modela kao i generisanje *CIM* profila na osnovu izabranih klasa i atributa. Za praćenje različitih verzija *CIM* modela definisati *ER* šemu, a podatke o pojedinim verzijama *CIM* modela smestiti u bazu podataka.

SPISAK KORIŠĆENIH SKRAĆENICA

CIM – Common Information Model

IEC – International Electrotecnical Commission

EMS – Energy Management System

EPRI – Electric Power Research Institute

CCAPI– Control Center Application Programming Interface

UML– Unified Modeling Language

XML– eXtensible Markup Language

RDF– Resource Descriptor Format

RDFS– Resource Descriptor Format Schema

OWL – Web Ontology Language

OMG – Object Management Group

XMI – Metadata Interchange

URI– Uniform Resource Identifier

CPSM – The Common Power System Model

DEWG– Data Exchange Working Group

ISO– Independent System Operator

RTO – Regional Transmission Organization

IOP – Interoperability

LINQ– Language Integrated Query

ORM– object - relational mapping

ORD– Object Relational Designer

DOM– Document Object Model

EA– Enterprise Architect

ER– Entity-relationship

EAP– Enterprise Architect Project

IDE– Integrated Development Environment

CLR –Common Language Runtime

CLI– Common Language Infrastructure

CIL– Common Intermediate Language

IL – Intermediate Language

W3C– World Wide Web Consortium

NERC – North American Electric Reliability Corporation

SADRŽAJ

1. UVOD

Još od pojave električne energije, proizvođači su nastojali da usavrše njen način proizvodnje kako bi ostali konkurentni na tržištu. Usavršavanje se odnosilo i na softversku podršku za elektroenergetske sisteme, za njihovu vizualizaciju i upravljanje. Za praćenje elektroenergetskih sistema korišćen je softver koji je najčešće kupovan od softverskih kuća ili je bio implementiran od samih elektroenergetskih kompanija.

Razvojem elektroenergetskih sistema postojala je potreba za čuvanjem i razmenom informacija u okviru same kompanije kao i između različitih kompanija. Aplikacije koje su razmenjivale informacije su najčešće bile nekompatibilne, rukovale su sa različitim formatima podataka, tako da je njihova komunikacija bila vrlo komplikovana. Da bi se omogućila nesmetana razmena informacija između aplikacija potrebno je ispoštovati određeni standard ili grupu standarda koje će aplikacije da koriste.

Jedna od važnih informacija koje se razmenjuju u ovoj komunikaciji, jeste model elektroenergetske mreže. Standard koji propisuje elemente i strukturu takvog modela se naziva *CIM (Common Information Model).* Upravo drugo poglavlje govori o ovom standardu. Kompanije, koje razvijaju sisteme za upravljanje prenosom električne energije, se u svom radu oslanjaju na *CIM*, a samim tim i na softverske alate koji omogućuju manipulacije *CIM* podacima. U radu je opisan softverski alat koji se bavi manipulacijom *CIM* podacima.

CIM model prati razvoj organizacije, te je sklon izmenama i javlja se potreba za čuvanjem *CIM* modela, ali sa svim izmenama koje su nad njih obavljene od početka pa do njegove krajnje realizacije. Da bi se lakše upravljalo različitim verzijama *CIM* modela, potrebno ih je čuvati u određenom formatu. Jedan od najprikladnijih načina jeste relaciona baza podataka. U ovom radu je realizovan softverski alat koji omogućava čuvanje *CIM* modela različitih verzija kao i generisanje *CIM* modela/profila u *RDFS* format podataka tj. format podataka koji je zgodan za razmenu informacija između aplikacija.

Treće poglavlje opisuje tehnologije koje su korišćenje u realizaciji softverskih alata koji će biti opisani.

U četvrtom poglavlju su prikazana dva softverska alata, realizovana u ovom radu, njihov grafički interfejs kao i opis njihove funkcionalnosti.

Peto poglavlje se odnosi na detalje implementacije aplikacija, sa akcentom na određene specifičnosti koje su realizovane u ovom radu.

U poslednjem, šestom poglavlju je dat kratak osvrt na rešenje problema uz ukazivanje na njegove dobre i loše strane kao i pravce daljeg istraživanja.

2. CIM

Common information model (*CIM*) [2] je apstraktni model koji predstavlja sve bitne objekte elektroenergetskog sistema. Koristeći jednostavni model, preduzeća i prodavci mogu smanjiti troškove integracije, koji treba da omogući veću funkcionalnost za upravljanje i optimizaciju električnih sistema [1]. Ovaj model uključuje opis svih klasa i njihovih atributa, kao i međusobnih veza između različitih klasa. Razvila ga je radna grupa 13 tehničkog komiteta 57 Međunarodne elektrotehničke komisije (*IEC – International Electrotecnical Commission, Tehnical Committee 57, Work Group 13*) i postao je međunarodni standard sa oznakom *IEC 61970-301*. Osnovna namena *CIM-a* je da olakša integraciju postojećih sistema za upravljanje prenosom, *EMS* (*Energy Management System*). Pre nastanka *CIM*-a, integracija sistema i razmena podataka između sistema različitih elektroenergetskih kompanija je bila jako otežana. Svaka kompanija je koristila svoj, nestandardni, model podataka i imala sopstveni način njegovog skladištenja. Ovakav način rada je kupce softverskih proizvoda za elektroenergetske sisteme primoravao da kupuju celokupna rešenja od istog proizvođača, dok je izgradnja sistema od više softverskih modula različitih proizvođača bila nezamisliva. Takva situacija je navela *EPRI* (*Electric Power Research Institute*) u Severnoj Americi da započne projekat pod nazivom *CCAPI* (*Control Center Application Programming Interface*). Cilj *CCAPI* projekta je bio razvoj standarda koji bi olakšali integraciju *EMS* aplikacija različitih proizvođača. Rezultat projekta je grupa standarda *IEC 61970*, čiji deo je i *IEC 61970-301* standard koji se odnosi na modele elektroenergetskih sistema [20].

CIM je definisan tehnikama za objektno-orijentisano modelovanje. Specifikacija *CIM-a* je data u *UML* (*Unified Modeling Language*) [3] notaciji. Modelovanje sistema pomoću *UML-a* obezbeđuje standardizovan način za predstavljanje resursa elektroenergetskog sistema pomoću klasa, njihovih atributa, i međusobnih veza između klasa. Način na koji su modelovani neki od osnovnih koncepata *CIM-a* je izložen u nastavku ovog poglavlja. Model podataka dat u obliku *UML* notacije nije pogodan za čitanje, odnosno obradu, od strane aplikacija. Rešavanjem ovog problema se bavi standard 501 [19] iz *IEC 61970* grupe standarda, koji definiše pravila za formiranje mašinski čitljivog formata *CIM-a*.

2.1 Osnove CIM-a

CIM se naslanja na više osnovnih tehnologija. Ovo poglavlje opisuje osnovne karakteristike tih tehnologija. Biće ukratko opisane svaka od njih, dovoljno da se shvate osnovni koncepti *CIM*-a.

Sledeće tehnologije su osnova za razumevanje osnova *CIM-a :*

- *Unified Modeling Language (UML)*
- *eXtensible Markup Language (XML)*
- *Ontologies*
- *Resource Descriptor Format (RDF)*
- *Web Ontology Language (OWL)*

2.1.1 Unified Modeling Language (UML)

UML je formalni opisni jezik koji objedinjuje nekoliko metodologija najčešće korišćenih od strane softver inženjera za modelovanje sistema. To je jezik a ne samo tehnika za crtanje dijagrama. Koristi se za definisanje, vizuelizaciju, kreiranje i dokumentovanje softverskih sistema. *UML* je zvanično definisan od strane *Object Management Group* (*OMG*) i zvanično je uključen u međunarodni standard [4].

UML dijagram je grafička reprezentacija modela sistema. Čitav model ne sadrži samo dijagrame već i pisane dokumente.

UML dijagrami se koriste da obezbede tri različita pogleda modela:

- Funkcionalni zahtevi
- Statička struktura
- Dinamički izgled

UML [1] modeli mogu da se razmenjuju između *UML* alata i softverskih sistema koristeći *XML Metadata Interchange (XMI)* format podataka. Metapodaci obezbeđuju "podatke o podacima " (tj. opisuju karakteristike i kontekst podataka u informacionom modelu). Metapodaci sadrže tehničke karakteristike, sadržaj i prirodu informacija entiteta, veze između entiteta i veze sa drugim podacima.

UML definiše notaciju i sintaksu za trinaest različitih tipova dijagrama, koji su kategorisani u tri grupe:

Strukturni dijagrami:

- Dijagram klasa (*Class Diagram*)
- Dijagram komponenti (*Component Diagram*)
- Objektni dijagram (*Object Diagram*)
- Kompozitni strukturni dijagram (*Composite Structure Diagram*)
- Dijagram rasporeda (*Deployment Diagram*)
- Dijagram paketa (*Package Diagram*)

Dijagrami zasnovani na izgledu:

- Dijagram aktivnosti (*Activity Diagram*)
- Dijagram slučajeva korišćenja (*Use Case Diagram*)

Dijagrami interakcije:

- Dijagrami sekvence (*Sequence Diagrams*)
- *Interaction Overview Diagram*
- Dijagrami za komunikaciju (*Communication Diagram*)
- Dijagrami zasnovani na vremenu (*Timing Diagram*)

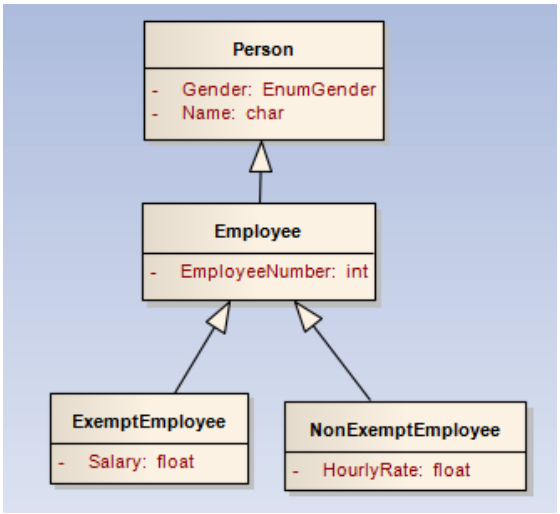
Radi boljeg razumevanja koncepata *CIM*-a u nastavku će biti objašnjeni dijagrami klasa. Dijagrami klasa vizuelno predstavljaju hijerarhiju klasa i relacija. Hijerarhija klasa je model sistema koji predstavlja svaku komponentu kao odvojenu klasu. Poštujući uobičajene principe modelovanja, hijerarhija klasa bi trebala da predstavi realnu strukturu sistema.

UML dijagrami klasa mogu biti podeljeni u odvojene grupe klasa. Takve grupe klasa predstavljene su paketima. U dijagramima klasa, one su grupisane po direktorijumima. Svaki paket ima svoje ime koje opisuje grupu klasa koje se nalaze u tom paketu. Kao sadržaj direktorijuma u fajl sistemu, klase u direktorijumima bi trebale biti povezane, i različiti paketi bi trebali biti opisani na način koji bi pomogao onome ko ih čita da razume grupisanje.

Klase (*Classes*) predstavljaju specifične tipove stvari koje se modeluju. U *UML* dijagramu klasa, klase su raspoređene po dijagramima kao kutije (*boxes*) koje imaju ime klase na samom vrhu. Svaka klasa pripada posebnom paketu. Jedan od uspeha pri modelovanju, jeste izbor prave grupe klasa, koje će se malo menjati pod uticajima novih zahteva, koji se postavljaju.

Jedan od načina da se smanji uticaj promena na sistem je da se koristi koncept pod nazivom generalizacija ili nasleđivanje (*inheritance*). U *UML*-u nasleđivanje se predstavlja linijom sa strelicom koja polazi od klase koja nasleđuje, a završava se na klasi koju nasleđuje data klasa.

Klase koje nasleđuju neku klasu se nazivaju deca (*child classes)*, dok se klasa koju nasleđuju zove roditeljska (*parent class*). Klase koje nemaju decu se nazivaju listovi (*leaf*).



Slika 2.1.1.1 Primer nasleđivanja klasa sa njihovim atributima

Klase poseduju svoje elemente koji se nazivaju atributima (*attributes*). Svaka klasa poseduje više instanci klase koje se nazivaju objektima (*objects*). Svaki objekat ima isti broj i iste tipove atributa, ali svaki atribut ima svoju internu vrednost za svaki objekat. U *UML* dijagramima klasa, atributi svake klase su predstavljeni u okviru date klase. Svaki atribut je opisan svojim imenom i tipom. Ono što treba napomenuti jeste da nisu svi atributi klase koji je opisuju prikazani u kutiji. Ukoliko neka klasa nasleđuje neku drugu klasu, atribute koje nasleđuje iz nje se ne prikazuju u njoj. Na slici 2.1.1.1 se može videti primer dijagrama klasa na kome se vidi nasleđivanje, kao i atributi klasa koje su predstavljene na dijagramu.

Klase takođe poseduju i veze koje opisuju kako je objekat povezan sa nekim drugim objektom. Ove veze se nazivaju asocijacijama (associations) u UML-u. Kao i atributi, svaka instanca klase ima isti broj i tip asocijacija ali sa svojim internim vrednostima. Postoje tri vrste asocijacija. Prva se naziva jednostavna asocijacija (simple association), druga je specijalan tip asocijacija koji se naziva agregacija (aggregation), i traća vrsta se naziva kompozicija (composition).

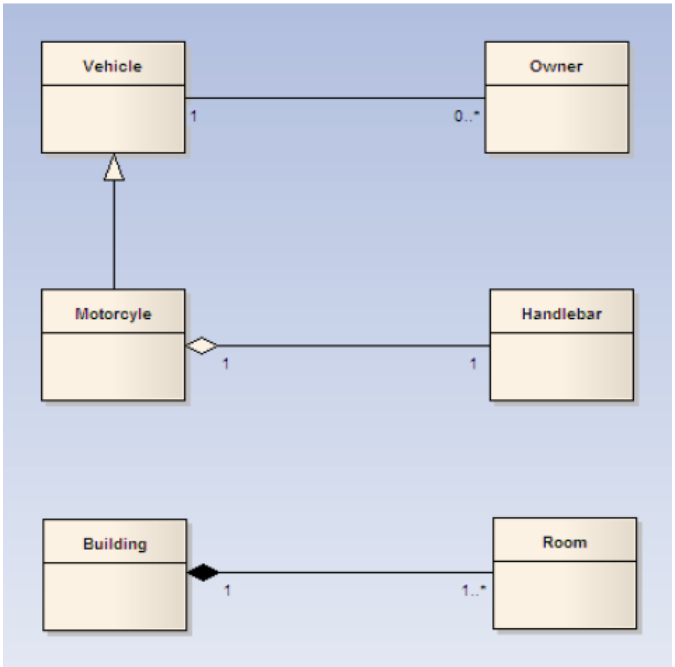
Jednostavna asocijacija se koristi za modelovanje udruživanja dve ravnopravne klase (klase na istom nivou). Ona je u UML-u predstavljena kao obična linija koja spaja dve klase (klasa **Vehicle** i **Owner**, slika 2.1.1.2)

Agregacija je vrsta asocijacije koja modeluje odnos “celina-delovi”, pri čemu su delovi samostalne klase čije instance mogu da “žive” i nezavisno od klase koja ih agregira. Ona je u *UML*-u predstavljena kao obična linija koja se na jednom kraju završava praznim rombom (klase **Motorcyle** i **HandleBar**, slika 2.1.1.2).

Kompozicija je vrsta asocijacije koja takođe modeluje odnos “celina-delovi”, pri čemu su delovi slabi objekti koji ne mogu samostalno da postoje bez osnovne klase, odnosno, prilikom oslobađanja osnovne klase, potrebno je osloboditi i delove. Ona je u *UML*-u predstavljena kao obična linija koja se na jednom kraju završava ispunjenim rombom (klase **Bulding** i **Room**, slika 2.1.1.2).

Asocijacije poseduju osobine (*properties*) koji reprezentuju broj mogućih veza između objekta i objekata sa kojim je taj objekat u vezi. Ta osobina se naziva kardinalitet (*multiplicity)*.

Kardinalitet je predstavljen u *UML*-u kao broj ili par brojeva na svakom kraju linije koja predstavlja vezu. Broj predstavlja minimum i maksimum mogućih veza. Na primer broj 1 znači da veza, od objekta ka objektu sa kojim je dati objekat spojen, može biti jedna i samo jedna. Na slici 2.1.1.2 je predstavljen dijagram klasa sa svim tipovima veza koje su opisane.



Slika 2.1.1.2 Primer dijagrama sa vezama između klasa

2.1.2 eXtensible Markup Language (XML)

XML je skraćenica za *Extensible Markup Language*, odnosno proširivi (metajezik) za označavanje (*markup*) tekstualnih dokumenata. *XML* je standardizovan jezik za čiju se standardizaciju brine *W3C*. Ideja je bila da se stvori jezik koji će i ljudi i računarski programi moći jednostavno da čitaju. *XML* definiše opštu sintaksu za označavanje podataka pomoću odgovarajućih tagova (*tags*) koje imaju poznato ili lako razumljivo značenje. Format koji obezbeđuje *XML* za računarske elemente može se prilagoditi najrazličitijim oblastima, kao što su elektronska razmena podataka, čuvanje podataka, odvajanje podataka od prezentacije, vektorska grafika, sistemi glasovne pošte, izrada novih specijalizovanih jezika za označavanje.

XML omogućava stvaranje dugotrajnih formata podataka koji su nezavisni od platforme. Često se dokumenti pisani na jednoj platformi ne mogu čitati na drugim platformama, niti u različitim programima na istoj platformi, čak ni u ranijoj verziji jednog programa na istoj platformi. U *XML* dokumentima tekstualni su ne samo podaci već i tagovi koji su smešteni u samoj *XML* datoteci. To znači da ih može čitati svaki alat koji je u stanju da čita tekstualne datoteke. Tako se podaci mogu prenositi sa jednog sistema na drugi.

Za potrebe *CIM*-a, *XML* se može upotrebiti za definiciju ontologija, uključujući *Resource Description Framework* (*RDF*) i *Web Ontology Language* (*OWL*), koji će biti opisani u narednim poglavljima [6].

2.1.3 Ontologies

Ontologija je definisanje pojmova i njihovih veza u okviru određene oblasti. Domen se može posmatrati kao jedno polje, ili oblast specijalnosti, poput elektrotehnike. Ontologija predstavlja posebno značenje termina u zavisnosti na koji se domen primenjuje.

Definicija koncepta ontologije počinje sa definicijom klase, atributa i veze koje mogu biti definisane u *UML* dijagramu, ali ide i dalje i definiše sledeće dodate pojmove:

- Ograničenja : formalni opis šta mora biti ispunjeno da bi za neke tvrdnje bio prihvaćen ulaz
- Pravila: *if-then* iskaz
- Aksiome: uslove i pravila koji zajedno čine ukupnu teoriju u domenu
- Događaji: menjanje atributa ili veza.

Dodatne konstrukcije, kao što su pravila i ograničenja čine korišćenje ontologija bogatijim alatom za definisanje modela podataka. Ontologije se najčešće kodiraju korišćenjem ontologije jezika. *RDF* i *OWL* koje će biti opisane u naredna 2 poglavlja su dve ontologije jezika koje se koriste u *CIM* standardu i *CIM* softverskim alatima [1].

2.1.4 Resource Descriptor Format (RDF)

RDF [1] je način definisanja modela podataka koji je opisan u *W3C* specifikaciji. *RDF* je zasnovan na ideji davanja iskaza o resursima u obliku "subjekat – predikat – objekat" izraza. Ovakav izraz se u *RDF* terminologiji naziva "tripletom" ("***triple***"), gde subjekat predstavlja resurs, objekat označava svojstvo ili atribut koji se pripisuje subjektu, dok predikat izražava vezu između subjekta i objekta.

Subjekat, odnosno resurs, je imenovan svojim jedinstvenim identifikatorom, *URI*-jem (*Uniform Resource Identifier*). *URI* je tekstualna vrednost koja jedinstveno identifikuje dati resurs. Mada je uobičajeno da *URI* resursa referencira stvarne podatke na *World Wide Web*-u, u slučaju *RDF*-a, on ne mora obavezno da predstavlja referencu za pristup resursu. Objekat i predikat "tripleta" takođe predstavljaju resurse. Ovakvim "tripletima" mogu da se formiraju sintaksne konstrukcije oblika "automobil ima četiri točka" ili "čovek ima ime". Kolekcija *RDF* iskaza predstavlja označen, usmeren multi-graf. *RDF* model podataka je, za reprezentaciju određenih vrsta znanja, pogodniji od relacionog modela ili drugih ontoloških modela .

Potrebno je napomenuti da se *RDF* model može izraziti u dva formata: *XML* i *N3* (*Notation 3*). *XML* format se jednostavno naziva *RDF*-om, jer je uveden kao deo *W3C* specifikacija koje definišu *RDF*. Pri tome, *XML* format ne treba poistovetiti sa apstraktnim *RDF* modelom. Drugi, *N3* format, je uveden za potrebe lakšeg pisanja i razumevanja podataka, jer njegova tabelarna notacija pruža bolju preglednost *RDF* "tripleta"od *XML* zapisa. *CIM* zajednica takođe koristi *Turtle* format (*Terse RDF Triple Language*) koji predstavlja minimalnu (sažetu) formu *N3* formata.

RDF Schema (*RDFS*) fajlovi opisuju klase, atribute i relacije informacionog modela i najčešće koriste *.rdfs* format. *RDF* fajlovi koji predstavljaju instance opisuju instance objekata i obično koriste *.xml* ili *.rdf* format.

U okviru *RDF* ili *RDFS* fajlova se koriste sledeći objekti:

Šema(Schema) fajl

- *Class*: Koristi se u *RDF* šemi za definisanje nove klase
- *Resource*: Korenska klasa za sve resurse
- *Property*: Klasa svih svojstava
- *Datatype*: Identifikuje tipove podataka
- *subClassOf*: Specijalizuje klasu

- *range*: Ograničava vrednosti svojstava
- *type*: Identifikuje klasu pojedinačnog resursa
- *about*: Opisuje postojeći resurs
- *Description*: Koristi se za osobine/vrednosti koje se odnose na resurs

Fajl koji predstavlja instancu

- *ID*: Identifikuje novi resurs
- *about*: Opisuje postojeći resurs
- *Description*: Koristi se za osobine/vrednosti koje se odnose na resurs

Najznačajnije prednosti *RDFS*-a se ogledaju u mogućnosti proširivanja opisa resursa novim svojstvima, i u slobodi izbora tehnike kojom će značenje resursa nekog domen-specifičnog rečnika biti opisano.

RDF šema može se proširiti da podrži neke od koncepata *UML*-a koji su značajni za korišćenje u *CIM*-u. Postoje sledeća proširenja *RDFS*-a:

Kardinalitet veze (*Myltiplicity*)

Jedna od značajnih informacija koje *UML* [19] notacija obezbeđuje, jeste kardinalitet uloge u asocijaciji između dve klase. *RDFS* se proširuje definisanjem novog svojstva, *cims:multiplicity,* i četiri imenovana resursa, koji, respektivno, predstavljaju kardinalitet uloge, odnosno moguće vrednosti kardinaliteta sa nazivima: M:0..1 (najviše jedan), M:1..1 (tačno jedan), M:0..n (nula ili više) i M:1..n (barem jedan). Mada za atribut klase nije uobičajeno razmatrati kardinalitet, pri RDFS definisanju atributa, može da se koristi ovde opisano svojstvo kako bi se naglasilo da li je dati atribut klase opcion (M:0..1) ili obavezan (M:1..1).

Inverzna uloga veze (*Inverse Rolename*)

Pošto asocijacija između dve klase ima dve strane tj. uloge, da bi se u okviru jedne obezbedio podatak za navigaciju do suprotne uloge, za resurs se definiše novo svojstvo *cims:inverseRoleName*, kojim se određuje suprotna strana asocijacije.

Sadržavanje (*isAggregate*)

Neke od asocijacija u *CIM*-u su specificirane kao agregacije, odnosno kompozicije. Definisanjem svojstva *cims:isAggregate* i postavljanjem vrednosti na *true*, ono definiše resurs kao kontejnersku ulogu asocijacije.

Stereotip (*Stereotype*)

CIM koristi *UML* stereotipove da označi da su neke klase primitivni tipovi ili enumeracije. Definisanjem svojstva *cims:stereotype* se iskazuje stereotip funkcije. Pored ove primarne namene, data konstrukcija se u praksi koristi da bliže označi i druge tipove resursa (ne samo klase).

Tipovi podataka (*Data type*)

Svaki atribut *UML* klase ima definiciju svog tipa. U *CIM* modelu, tipovi podataka su definisani kao klase. Definisanje tipova podataka atributa klase se referenciraju na korespondentne tipove podataka klasa. Tipovi podataka mogu biti*: string, enumeration, float, integer* i drugi. Tip podataka se opisuje svojstvom *cims:dataType*.

Profil (*Profile*)

Deo klasa, atributa i asocijacija u *UML* modelu mogu biti definisani kroz profil. Svaki profil poseduje ime. Profil se opisuje sa *cims:profile* svojstvom. Kao vrednost ovog svojstva navodi se ime profila.

Kategorija klase(*Class category*)

CIM UML model klasa koristi kategorije za strukturiranje klasa. Kategorije su npr: **Core** ili **Domain**. Svojstvo *cims:ClassCategory* definiše kategoriju. Ime kategorije je definisano u *rdf:about* svojstvu.

Pripadnost kategoriji (*Belongs To Category*)

Svaka klasa u *UML* modelu pripada nekoj kategoriji. Svojstvo *cims:BelongsToCategory* se referencira na kategoriju klase kojoj pripada.

2.1.5 Web Ontology Language (OWL)

OWL je jezik zasnovan na *RDF-u* koji je sposoban da u potpunosti izražava ontologije, za razliku od *UML* dijagrama klasa ili standardnog *RDF-a*. *OWL* je namenjen kada je informacije sadržane u dokumentima potrebno obraditi od strane aplikacije, za razliku od situacija u kojima sadržaj treba da bude predstavljen ljudima [7]. *OWL* se može koristiti za eksplicitno predstavljanje značenje reči u rečenicama i odnosa između pojmova u rečenici. *OWL* ima više mogućnosti za izražavanje značenja i semantike nego *XML, RDF* i *RDFS*, a time prevazilazi ove jezike za razliku od kojih može da predstavi mašinsku interpretaciju sadržaja na internetu.

OWL koriste neki *CIM* alati da odrede podskupove celog modela podataka, koji se nazivaju profili (opisano u poglavlju 2.2). *CIM* model podataka je definisan u *UML-u* i najčešće se razmenjuju korišćenjem *XMI-a*.

Zvanično, *CIM* standard opisuje profile samo kao tekstualne dokumente. Međutim, upotreba na engleskom jeziku, kao definicija sintakse u ovim dokumentima ograničava preciznost i zahteva nekoga da tumači jezik. Najverovatnije da će zvanični standard pratiti neki od *CIM* alata i biti u mogućnosti da opiše *OWL* u bliskoj budućnosti [1].

2.2 *CIM* profil

Profil (*profile*) se definiše kao skup klasa, njihovih atributa i međusobnih veza koji predstavlja podskup svih klasa, atributa i veza definisanih u nekoj postojećoj šemi. Dakle, dati profil predstavlja podskup sebi nadređene šeme. Obično se svakom profilu dodeljuje naziv i određeni imenski prostor (*namespace*). Profili se koriste za definisanje kontekst-zavisnih ili domenskih modela. Profili mogu da se realizuju u različitim formatima, kao što su *RDF* , *XML Schema*, tekst dokument ili *HTML* [1].

CPSM profil (*The Common Power System Model*) definiše podskup klasa, njihovih atributa i međusobnih veza neophodnih za izvršavanje elektroenergetskih *EMS* aplikacija. *CPSM* profil je originalno definisan od strane *NERC*, radne grupe *DEWG*, ali ga je *IEC* kasnije proširio. *CPSM* profili se uobičajeno koriste za razmenu modela prenosnih mreža između severnoameričkih *ISO* (*Independent System Operator*), odnosno *RTO* (*Regional Transmission Organization*) organizacija i drugih distributivnih i prenosnih kompanija. Upotreba *CPSM* profila je široko zastupljena, jer *NERC* zahteva njegovu upotrebu, pa se iz tih razloga ponekad naziva i *NERC* profilom.

CPSM profil je definisan u standardu *IEC 61970-452*. Osnovna namena ovog standarda je postizanje sledećih ciljeva: unapređenje preciznosti modela elektroenergetskih sistema korišćenih u kritičnim sistemima, postizanje konzistentnosti među modelima korišćenim od strane različitih sistema koji igraju ulogu u radu ili planiranju interkonekcije, smanjenje ukupne cene održavanja kritičnih modela koji se koriste u radu ili planiranju interkonekcije.

Testiranje *CPSM* profila se vrši kao deo godišnjih testova interoperabilnosti, *IOP* (*Interoperability*), koje sponzoriše *EPRI*. U okviru ovih testova se između velikih *EMS* kompanija razmenjuju uzorci modela i profili u *CPSM* formatu i zatim se rezultati upoređuju kako bi se utvrdilo jesu li modeli ispravno tumačeni od strane *EMS* aplikacija.

2.3 Mapiranje *CIM*-a iz *UML*-a na *RDFS*

Proširivanjem *RDFS* [19] jezika prethodno je omogućeno potpuno mapiranje *CIM* modela predstavljenog u *UML* notaciji, u njegovu mašinski-čitljivu *XML* reprezentaciju korišćenjem *RDFS*-a. (poglavlje 2.1.4)

UML paketi se mapiraju na *cims:ClassCategory* resurs, klase se mapiraju *na rdfs:Class* resurse. Atributi se mapiraju na *rdf:Property* resurse. Vrednost enumeracije se mapiraju na *rdf:Description* resurs. Uloga asocijacije se mapira na *rdf:Property* resurs.

Za resurs dobijen iz paketa, definišu se svojstva: *rdfs:label* i *rdfs:comment*.

Za resurs dobijen iz klase, definišu se svojstva*: rdfs:label, rdfs:comment, rdfs:subClassOf, cims:stereotype* i *cims:belongsToCategory*.

Za resurs dobijen iz atributa, definišu se svojstva: *rdfs:label, rdfs:comment, rdfs:domain* i *rdfs:range*.

Za resurs dobijen iz uloge asocijacije, definišu se ista svojstva kao i za atribut, kao i dodatna svojstva: *cims:inverseRoleName, cims:multiplicity i cims:isAggregate*.

Za resurs dobijen iz vrednosti enumeracije, definišu se svojstva: *rdfs:label, rdfs:comment* i *rdf:type*.

2.4 Struktura CIM-a

2.4.1 Osnovni paketi *CIM*-a

CIM je definisan objektno-orijentisanim modelom u *UML* notaciji, koja *CIM* definiše kao skup paketa (Slika 2.4.1.1). Paketi predstavljaju način za grupisanje srodnih, logički povezanih klasa, tako da olakšavaju razumevanje i korišćenje modela. Klase opisuju elektroenergetski sistem, tj. proizvodnju električne energije, njen prenos i distribuciju. Paketi su dalje organizovani u grupe, čije specifikacije predstavljaju zasebne standarde.

U daljem tekstu biće navedeni osnovi paketi CIM modela kao i klase koje im pripadaju.

Core paket sadrži osnovne klase koje su potrebne EMS aplikacijama, kao što su *IdentifiedObject, PowerSystemResource, EquipmentContainer, ConductingEquipment,* i *Terminal* . *Core* paket ne zavisi od ostalih paketa, ali većina ostalih paketa su zavisni od njega.

Topology paket proširuje navedeni *Core* paket, tako što zajedno sa *Terminal* klasom modeluje konektivnost, tj. fizičku povezanost opreme - izgrađenost mreže. Osim toga on modeluje i topologiju, logičku povezanost opreme preko zatvorenih prekidača.

Wires paket proširuje *Core* i *Topology* pakete, modelujući informacije o električnim karakteristikama prenosnih i distribucionih mreža.

Outage paket proširuje *Core* i *Wires* pakete i modeluje informacije o tekućim i planiranim konfiguracijama mreže.

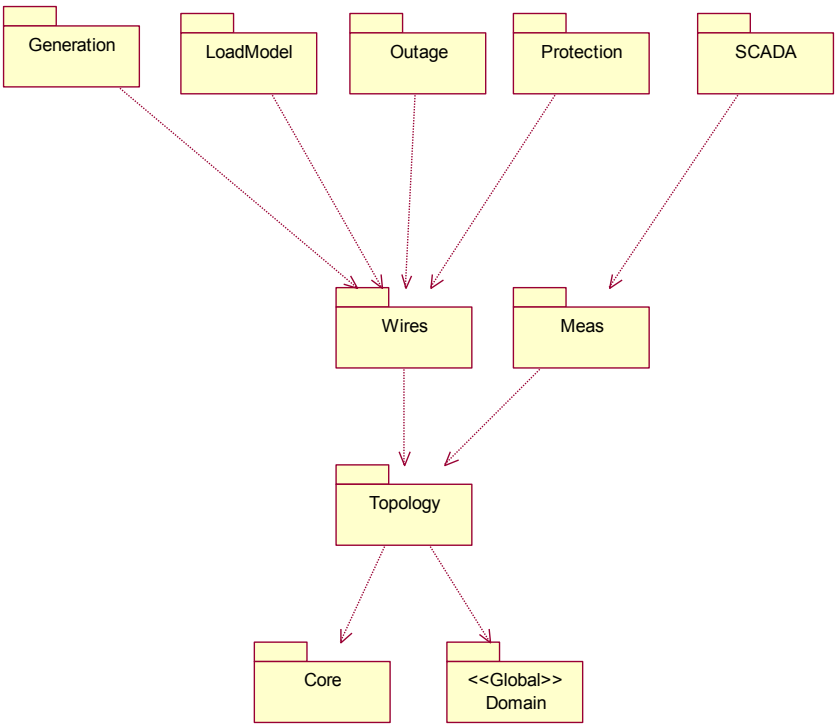
Protection paket proširuje *Core* i *Wires* pakete, i sadrži modele zaštitne opreme, poput releja.

Meas paket sadrži modele merne opreme.

LoadModel paket se sastoji od modela potrošača i krivih potrošača. Specijalne okolnosti koje mogu uticati na opterećenje, kao što su sezona, tip dana (ponedeljak, utorak, ... , praznik), su takođe uključeni u ovaj paket.

Generation se sastoji iz dva podpaketa *Production* i *GenerationDynamics* paketa. *Production* paket sadrži modele raznovrsnih generatora. Pored toga modeluje i cenu proizvodnje. *GenerationDynamics* paket sadrži modele pogonskih motora, kao što su turbine i bojleri.

SCADA paket sadrži modele svih podataka vezanih za daljinsko upravljanje.



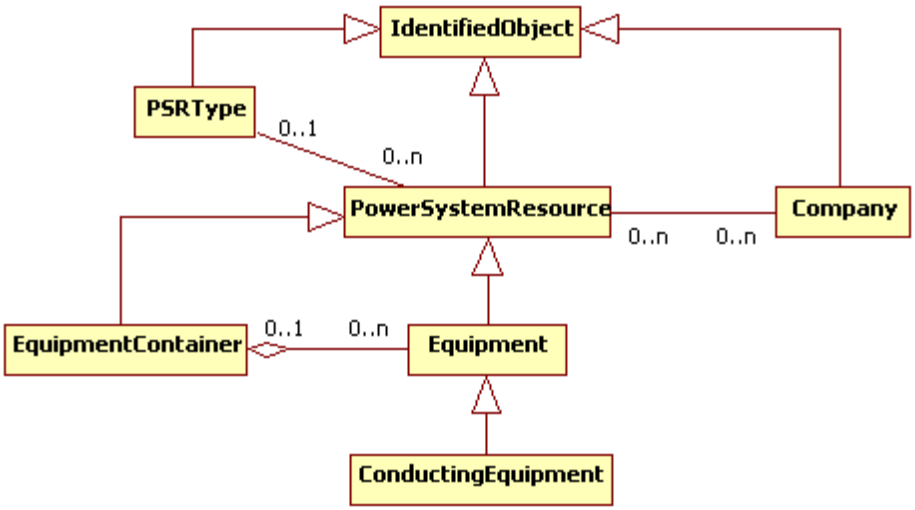
Slika 2.4.1.1. Struktura *CIM* modela

2.4.2 Osnovne klase *CIM*-a

Pod osnovnim klasama se podrazumevaju one klase koje modeluju osnovne vrste entiteta koji mogu da se nađu u okviru elektroenergetskog sistema modelovanog *CIM*-om. Dijagram odnosa ovih klasa je prikazan na slici 2.2.

Klasa *IdentifiedObject* se nalazi na samom vrhu hijerarhije. Iz nje je izvedena većina CIM klasa [20], a sadrži atribute za imenovanje entiteta. Klasa koja se smatra osnovnom, jeste *PowerSystemResource*. Ona predstavlja jedan resurs elektroenergetskog sistema. Resurs može da se nalazi u vlasništvu jedne ili više kompanija, gde je kompanija predstavljena klasom *Company*. Klasom *Equipment* je predstavljena sva provodna i ostala oprema elektroenergetskog sistema. Klasom *EquipmentContainer* su predstavljeni svi objekti koji na neki način grupišu opremu. Klasa *ConductingEquipment* predstavlja provodnu opremu, tj. deo opreme koja može da provodi električnu energiju, kao što su: prekidači, sabirnice, itd. Posebnu namenu u modelu ima klasa *PSRType*, čija glavna uloga je da klasifikuje primerke iste klase, odnosno da omogući manje nestandardne modifikacije *CIM*-a, bez potrebe da se menja model. Dakle, svi entiteti koji se mogu naći u okviru elektroenergetskog sistema modelovanog CIM-om, mogu da se podele u sledeće podskupove:

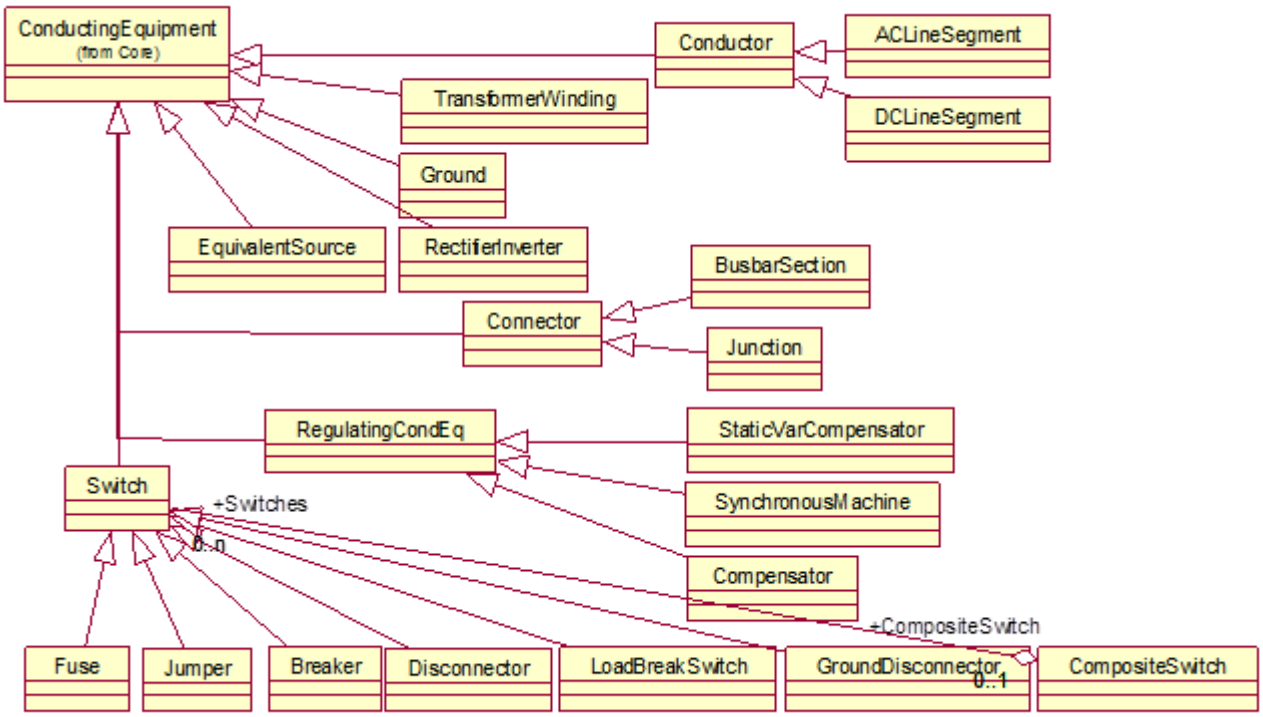
- provodna oprema – sve klase izvedene iz *ConductingEquipment*;
- bilo koja oprema – klase izvedene iz *Equipment*;
- entiteti za grupisanje, odnosno skladištenje opreme – klase izvedene iz *EquipmentContainer*;
- ostali tipovi – klase izvedene iz *PowerSystemResource*.



Slika 2.4.2.1 Osnovne klase *CIM* modela

2.4.3 Provodna oprema

Provodnu opremu u okviru *CIM*-a modeluju sve klase izvedene iz klase *ConductingEquipment*. Slika 2.4.3.1 prikazuje dijagram klasa provodne opreme, gde se lako može uočiti specijalizacija različitih tipova ove opreme.



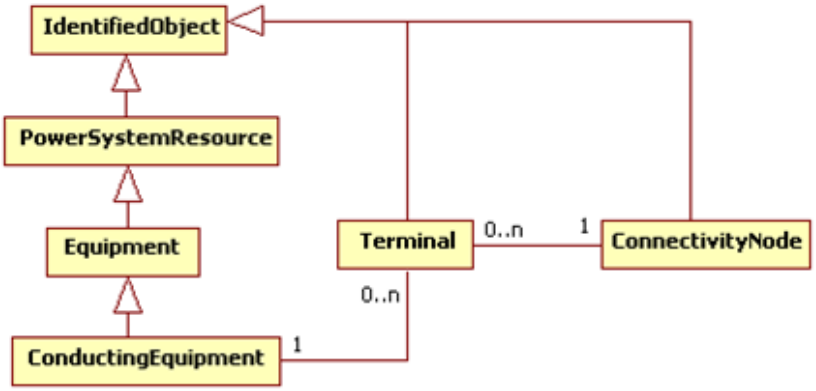
Slika 2.4.3.1. Provodna oprema *CIM*-a

2.4.4 Povezivanje provodne opreme

Za modelovanje elektroenergetskih sistema nije dovoljno predstaviti samo elemente koji postoje u sistemu, već u obzir moraju da se uzmu i električne veze između njih. Da bi *CIM* model elektroenergetskog sistema bio potpun, potrebno je obezbediti mehanizme za povezivanje provodne opreme. Konektivnost u sistemu *CIM* modeluje pomoću posebnih klasa: *Terminal* i *ConnectivityNode*. Ove klase ne modeluju resurse elektroenergetskog sistema, već imaju ulogu da obezbede informacije o fizičkoj povezanosti elemenata. Dakle provodna oprema u *CIM*-u se ne povezuje direktno, nego isključivo putem ove dve klase. Dijagram klasa prikazan na slici 2.4.4.1. ilustruje način na koji *CIM* omogućava povezivanje provodne opreme.

Klasa *ConductingEquipment*, koja predstavlja provodnu opremu, je u vezi sa klasom *Terminal*, koja predstavlja pristupne tačke preko kojih se provodna oprema povezuje u mrežu. Provodna oprema u mreži može da ima jedan ili više *Terminal*-a. Dva *Terminal*-a ima sledeća oprema: *ACLineSegment*, *DCLineSegment*, *Jumper*, *Fuse*, *Breaker*, *Disconnector* i *LoadBreakSwitch*, dok sva ostala provodna oprema ima jedan *Terminal*.

Da bi se obavilo logičko povezivanje provodne opreme, uvedena je klasa *ConnectivityNode*. Instance ove klase predstavljaju čvorove mreže i služe za grupisanje *Terminal*-a koji su povezani preko nulte impedanse. Na primer, ukoliko je u modeliranom sistemu elemenat A u direktnoj vezi sa elementom B, onda se njihova veza u *CIM*-u modeluje tako što se *Terminal* elementa A i *Terminal* elementa B povežu istim *ConnectivityNode*-om. Dakle, *ConnectivityNode* može da referencira više *Terminal*-a, dok *Terminal* može da referencira samo jedan *ConnectivityNode*.



Slika 2.4.4.1. Prikaz dela modela povezivanja opreme

3. OPIS KORIŠĆENIH TEHNOLOGIJA I ALATA

U ovom radu je korišćeno *Microsoft Visual Studio 2008* razvojno okruženje i *.NET Framework* . Za implementaciju aplikacija korišćen je C# programski jezik. Za čuvanje *CIM* modela (profila) korišćena je *Microsoft*-ova baza podataka (*Microsoft SQL Server 2005 Express Edition*). Za komunikaciju sa bazom korišćen je *LINQ*. Generisanje *Microsoft Excel* fajlova omogućeno je korišćenjem standardnih biblioteka *.NET Framework*-a, dok je komunikacija sa *Enterprise Arhitectom* omogućena preko *Automation Interface*-a tj. *Interop.EA.dll* fajla.

U daljem tekstu biće opisani neki od alata i tehnologija koje su korišćene u radu.

3.1 Microsof Visual Studio i *.NET Framework*

Microsoft Visual Studio 2008 je verovatno jedno od najnaprednijih integrisanih razvojnih okruženja (*Integrated Development Environment-IDE*) dostupno za programere danas. Bazirano je na dugoj istoriji programskih jezika i interfejsa [17].

Microsoft .NET Framework je platforma za razvijanje softvera fokusirana na objedinjavanju razvoja svih *Microsoft* aplikacija. Osnovne ideje su platformska nezavisnost i mrežna transparentnost. Ako je suditi po *Microsoft*-u, *.NET* uključuje brojne tehnologije dizajnirane da omoguće brzi razvoj interneta i intranet aplikacija.

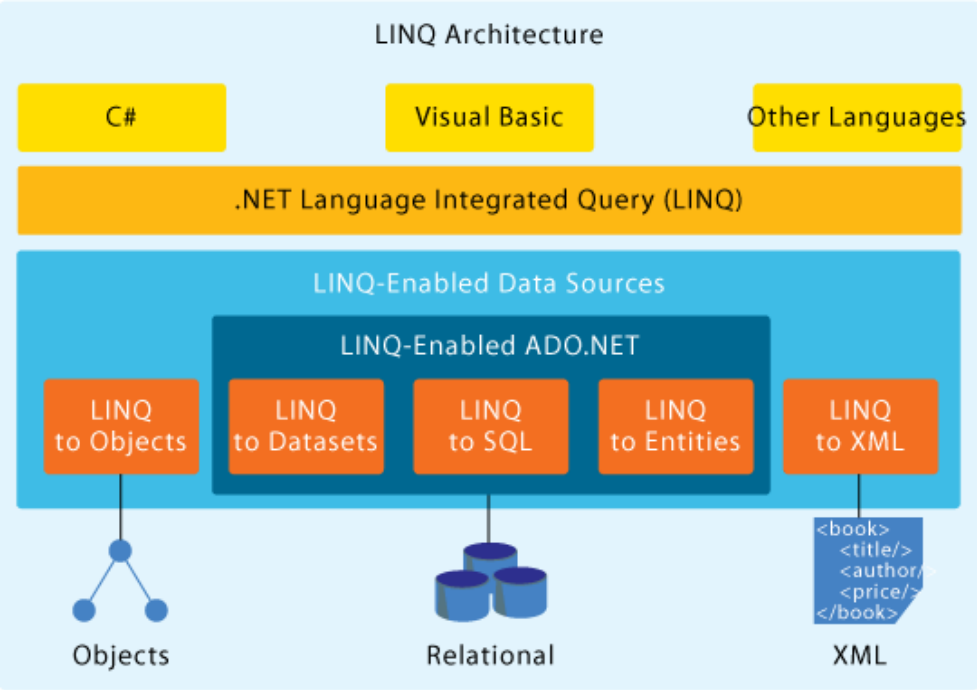
Dve glavne komponente *.NET-a* su *Common Language Infrastructure* (*CLI*) i *Common Language Runtime (CLR). CLI* je skup specifikacija za *runtime* okruženje, uključujući *common type system, base class library* i *machine-independent intermediate code* ili *Common Intermediate Language* (*CIL*). *CLR* obezbeđuje platformu za izvršavanje *CLI* koda; pre nego što *CIL* može biti izvršen, *CLR* ga mora prevesti (obično putem *just-in-time* kompajlera) u nativni mašinski kod.

Još jedan bitan deo *.NET-a* je i programski jezik stvoren za njega – *C#.* Za razliku od drugih jezika, on ne poseduje sopstvene biblioteke tj. nema '*stdio.h'* ili '*iostream'* niti išta slično. To je tako jer je osmišljen kao jezik koji bi koristio *.NET* biblioteke kao svoje sopstvene. Dok je, zbog ovoga, istovremeno *C#* vrlo zavisan od *.NET-a*, on sam nije deo *.NET*-a. Ne igra nikakvu specifičnu ulogu u *.NET* svetu - i on se kompajlira u *IL* (*Intermediate Language*) kao i svaki drugi - i ima slične mogućnosti kao i svi ostali jezici.

3.2 LINQ(Language Integrated Query)

LINQ predstavlja novi dodatak *C#* programskog jezika. Pojavljuje se u verziji *C# 3.0. LINQ* rešava problem manipulacije sa velikim kolekcijama objekata, gde je potrebno izdvojiti deo kolekcije tokom zadatka za koji se program izvršava.

Ranije, ovakva vrsta posla zahtevala je pisanje velike količine koda, kao što su sortiranje ili grupisanje, dok danas *LINQ* to sam odrađuje. *LINQ* obezbeđuje elegantan upitni jezik koji poseduje veliki broj funkcija koje omogućavaju brzo sortiranje, grupisanje i izračunavanje statistike nad rezultatima upita. *LINQ* takođe omogućava upite nad bazama podataka ili kompleksnim *XML* dokumentima u kojima treba manipulisati ili pretražiti milione ili čak milijarde objekata [8] . Tradicionalno, ovaj problem je uglavnom rešen sa specijalizovanim bibliotekama klasa, pa čak i pomoću različitih jezika, kao što su *SQL* upitni jezik za baze podataka.



Slika 3.1.1 Arhitektura *LINQ*-a

Na slici 3.1.1 je predstavljena arhitektura *LINQ*-a (preuzeto sa [12]).

Visual Studio 2008 dolazi sa nekoliko varijanti *LINQ*-a koji omogućava postavljanje upita za različite tipove podataka: *LINQ to Objects, LINQ to SQL, LINQ to XML*.

LINQ to Objects - Obezbeđuje upite nad bilo kojom vrstom C# objekata u memoriji, kao što su nizovi, liste, i ostale vrste kolekcija.

LINQ to SQL - Obezbeđuje upite nad relacionim bazama podataka koje koriste standardni *SQL* upitni jezik, kao što je *Microsoft SQL Server, Oracle* i drugi. Ranije, da bi pristupili uz pomoć *C#* ovim bazama podataka korisnik je morao da zna *SQL*, ali danas mogućnosti upitnog jezika su ugrađenje u okviru *C#* programskog jezika i kroz *LINQ* lako se dozvoljava *LINQ to SQL* da upravlja *SQL* prevodu za vas.

LINQ to XML - Obezbeđuje kreiranje i manipulaciju *XML* dokumentima koristeći istu sintaksu i opste mehanizme upita kao ostale varijante *LINQ*-a.

U nastavku teksta biće detaljnije objašnjene neko od varijanti *LINQ*-a koje su i korišćene ovom radu prilikom implementacije.

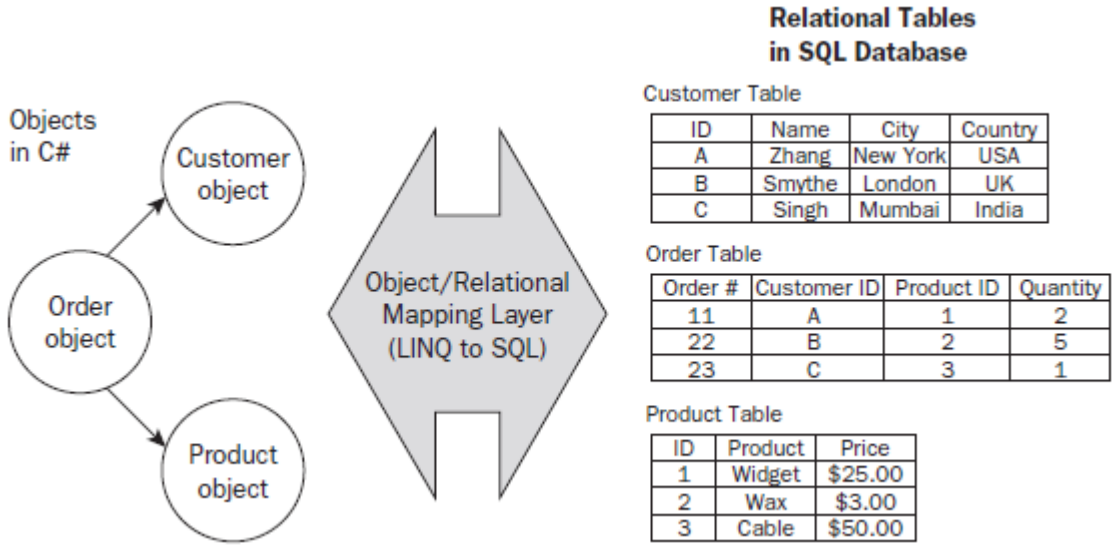
3.2.1 LINQ to SQL

LINQ to SQL [9] je komponenta *.NET Framework* koja obezbeđuje *run-time* infrastrukturu za mapiranje objekata na relacioni bazu podataka i obratno (eng. *object - relational mapping* (*ORM*)).

LINQ to SQL model podataka relacione baze mapira na objektni model predstavljen u programskom jeziku. Kada se aplikacija pokrene, *LINQ to SQL* vrši upite nad objektnim modelom, prevodi ih u *SQL* i šalje ka bazi podataka na izvršavanje. Kada baza podataka vrati rezultat, *LINQ to SQL* ga prevodi u objekte sa kojim programer lako može da radi u svom programskom jeziku.

Microsoft Visual Studio sadrži *Object Relational Designer* [10] koji služi za generisanje objektnog modela u aplikaciji, koji se mapira na objekte u bazi podataka. Takođe se generiše strogo

definisan ***DataContext*** [14] čija je namena slanje i primanje podataka između klasa i baze podataka.

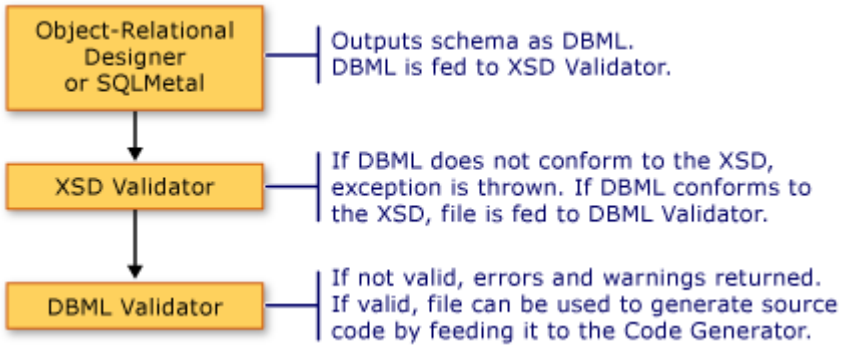


Slika 3.1.2 *ORM* mapiranje

Na slici 3.1.2 se može videti jedan primer *ORM* mapiranja. Sa leve strane slike može se videti objektni model tj. 3 objekta koja su međusobno povezana (***Customer object***, ***Order object***, ***Product object***). Sa desne strane se nalazi relacioni model (***Customer Table***, ***Order Table***, ***Product Table***). Središnji deo slike (*Object/Relational Mappint Layer*) predstavlja *LINQ to SQL* koji je posrednik u mapiranju iz objektnog modela u relacioni i obratno.

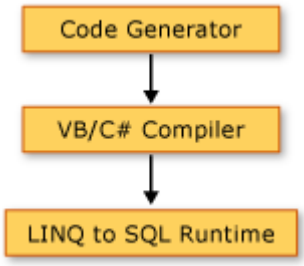
Ono što treba napomenuti jesu dve komponente koje su sastavni deo *LINQ to SQL* komponente [13]. To su *DBML Extractor* i *Code Generator*.

DBML Extractor preuzima metapodatke iz baze podataka kao ulaz i kreira *DBML* fajl na izlazu. Slika 3.1.3 može ilustrovati sekvencu operacija.



Slika 3.1.3 *DBML Extractor*

Code Generator prevodi *DBML* fajlove u *Visual Basic, C#* ili *XML* fajlove za mapiranje. Slika ispod može ilustrovati sekvencu operacija koje se izvršavaju.



Slika 3.1.4 *Code Generator*

Ukratko, dovoljno je da je kreirana šema baze, i na osnovu njenih metapodataka može da se izgeneriše *DBML* fajl koji se dalje prevodi u programski kod, u ovom radu konkretno *C#* programski kod. Taj kod ujedno predstavlja objektni model baze podataka.

3.2.2 LINQ to XML

Kao što je ranije napomenuto *LINQ to XML* služi za rad sa *XML* dokumentima. *LINQ to XML* ne postoji da zameni standardne *XML API*-je kao što su *XML DOM (Document Object Model), Xpath, Xquery, XSLT* i ostale. *LINQ to XML* dopunjava ove standarde *XML* klasama i omogućava lakši rad sa *XML* dokumentima. On daje vrhunske mogućnosti za kreiranje i postavljanje upita nad *XML* podacima, a rezultat je jednostavan kod i brz razvoj za mnoge jednostavne situacije, posebno ako je korisnik ranije koristio *LINQ to Objects ili LINQ to SQL* u ostalim delovima programa.

Prednosti *LINQ to XML* u odnosu na *XPath* su sto *XPath* ne dozvoljava projekciju novih tipova. On može samo da vrati kolekciju čvorova stabla, za razliku od koga *LINQ to XML* može da izvrši upit i projekciju objekata grafa ili *XML* stabla u novom obliku. *LINQ to XML* upiti su mnogo funkcionalniji i moćniji od *XPath* izraza. *C#* kompajler ne parsira *XPath* izraze u vreme kompajliranja. Sa druge strane, *LINQ to XML* upiti se parsiraju i kompaliraju od strane *C#* kompajlera. Kompajler je u mogućnosti da obradi veliki broj grešaka koje mogu da se dese prilikom postavljanja upita.

3.3 Enterprise Arhitect (EA)

Enterprise Arhitect (*EA*) [21] je alat za dizajn i analizu *UML*-a, koji pokriva sve aspekte razvojnog ciklusa softvera od prikupljanja zahteva, preko analize, projektovanja modela, testiranja, podrške pri implementaciji.

EA je baziran na poslednjoj verziji *UML 2.1* specifikacije (pogledati www.omg.org). Karakterišu ga visoko performanse, intuitivan interfejs koji omogućava napredni nivo modelovanja. To je više-korisnički vizuelni alat sa velikim skupom funkcija, koji pomaže analitičarima, ljudima koji testiraju sistem, projekt-menadžerima, ljudima koji vrše kontrolu kvaliteta. *UML* definiše vizuelni jezik koji se koristi za modelovanje jednog dela domena sistema (poglavlje 2.1.1).

3.3.1 Automation Interface

Automation Interface omogućava pristup unutrašnjosti *EA*-og modela. U nastavku su dati samo neki primeri zadataka koji se mogu obaviti korišćenjem *Automation Interface*:

- Obavljanje poslova koji se ponavljaju, kao što je numerisanje verzija elemenata u modelu
- Generisanje korisničkih izveštaja
- Izvršavanje *Ad hoc* upita nad modelom

Sva razvojna okruženja koja mogu da generišu *ActiveX Com* klijente trebali bi da budu u stanju da se povežu sa *Enterprise Architect Automation Interface*-om.

Dodaci *(add-ins)* omogućavaju dodavanje funkcionalnosti *Enterprise Arhitect*-u. Zahvaljujući njemu moguće je proširenje korisničkog interfejsa *EA*-a. Dodaci pružaju nekoliko ključnih prednosti u odnosu na samostalnu automatizaciju klijenta, uključujući mogućnost definisanja dodatnih *EA* menija i primanja obaveštenja o raznim *EA* događajima od strane korisničkog interfejsa, kao što su klik na menije ili selekcija od strane korisnika.

Takođe je omogućena integracija EA u okviru *Microsof Visual Studio 2008*. To omogućava korisniku prikaz i editovanje *UML* modela u okviru *Visual Studio* okruženja. Na taj način je omogućen veliki broj operacija *EA*-a koje su podržane u samom *Visual Studio* okruženju.

4. PRIKAZ REALIZOVANIH SOFTVERSKIH ALATA

U ovom poglavlju je dat opis softverskih alata koji su nastali kao posledice zahteva, koji su formulisani za njihovu realizaciju.

4.1 Formulacija zahteva

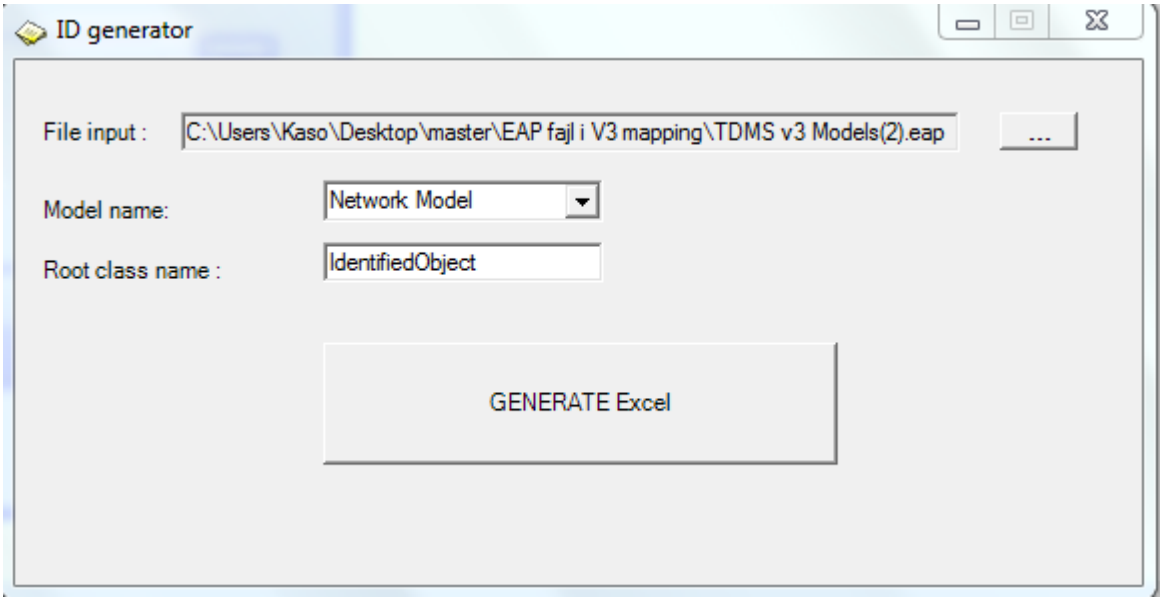
Osnovni zahtevi koji su postavljeni pre implementacije postojećih aplikacija su sledeći:

- Omogućiti generisanje konfiguracionog fajla, na osnovu izvornog *CIM* modela definisanog u *Enterprise Arhitect*-u, u *Excel* format, poštujući osnovni izgled generisanog fajla.
- Definisanje *ER (Entity-relationship)* šeme za smeštanje *CIM* modela (profila) definisanih u *Enterprise Arhitect*-u.
- Omogućiti brisanje modela (profila) iz baze podataka na zahtev korisnika.
- Omogućiti vizuelnu reprezentaciju modela (profila), i operacije dodavanja, izmene i brisanja nad elementima (klasa, atribut, veza) modela.
- Omogućiti čuvanje izmena koje su izvršene nad modelom (profilom) u već definisanu bazu podataka.
- Omogućiti generisanje *CIM* profila na osnovu izabranih klasa, atributa u *RDFS* format
- Omogućiti sinhronizaciju *CIM* modela (profila) smeštenog u bazi sa *CIM* modelom(profilom) definisanim u *EAP* fajlu.

Treba napomenuti da je prvi zahtev formulisan za aplikaciju *ID generator*, dok su preostali formulisani za aplikaciju *CIM manager.* U nastavku će biti više reči o navedenim aplikacijama.

4.2 Aplikacija *ID generator*

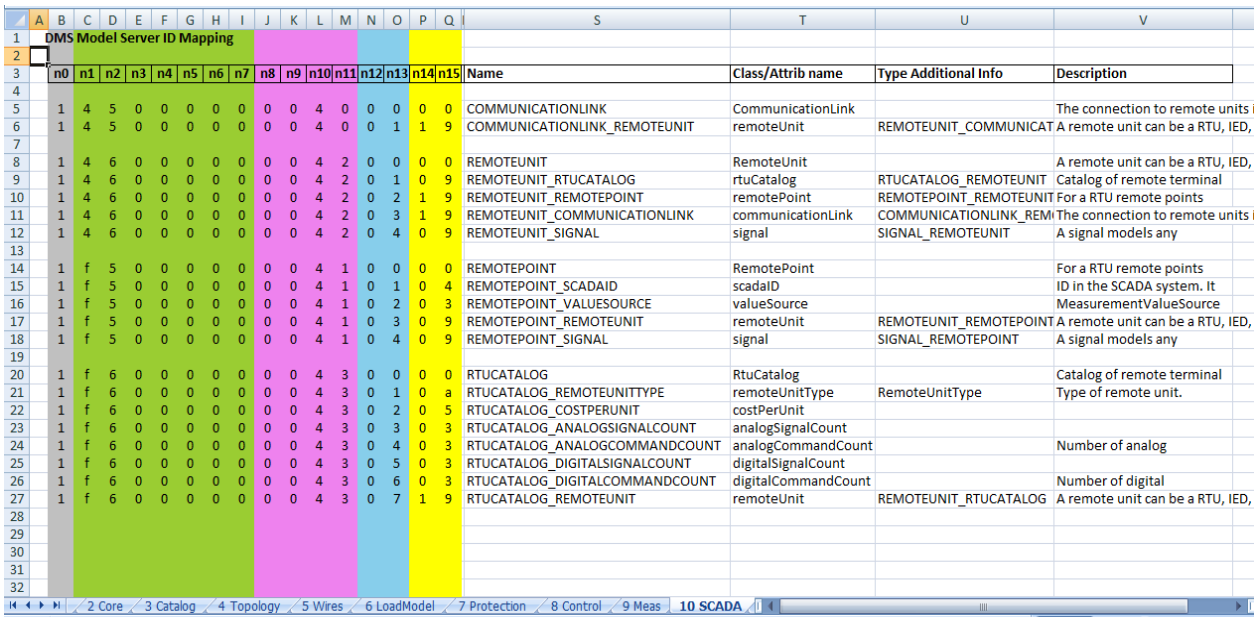
ID generator predstavlja aplikaciju koja služi za generisanje konfiguracionog fajla čija namena će biti opisana kasnije. Fajl se čuva u *Excel* fajlu (*.xls*). Aplikacija je realizovana u programskom jeziku *C#* kao *GUI* aplikacija pod *Visual Studio 2008* okruženjem.



Slika 4.2.1 Glavni prozor aplikacije ***ID generator***

Na slici 4.2.1 je prikazana glavna forma aplikacije. *File input* polje predstavlja putanju do *EAP* (*Enterprise Architect Project*) fajla, u kome se nalazi objektni model, zatim polje *Model name* predstavlja ime modela za koji želimo da generišemo konfiguraciju, kao i *Root class name* koje predstavlja klasu koja se za izabrani model nalazi na vrhu hijerarhije (klasa koja ne nasleđuje ni jednu drugu klasu). Pritiskom dugmeta *GENERATE Excel* posle nekog vremena kreira se konfiguracioni fajl čija je struktura prikazana na slici 4.2.2.

Svaka kartica (*eng. Sheet*) generisanog fajla predstavlja jedan paket *EAP* fajla. Svaka kartica sadrži klase odgovarajućeg paketa, a svaka klasa sadrži svoje atribute. U nastavku će detaljno biti objašnjena struktura generisanog fajla. Na slici 4.2.2 je prikazan konfiguracioni fajl ***SCADA*** paketa.



DMS Model Server ID Mapping

n0	n1	n2	n3	n4	n5	n6	n7	n8	n9	n10	n11	n12	n13	n14	n15	Name	Class/Attrib name	Type Additional Info	Description
1	4	5	0	0	0	0	0	0	0	4	0	0	0	0	0	COMMUNICATIONLINK	CommunicationLink		The connection to remote units
1	4	5	0	0	0	0	0	0	0	4	0	0	1	1	9	COMMUNICATIONLINK_REMOTEUNIT	remoteUnit	REMOTEUNIT_COMMUNICAT	A remote unit can be a RTU, IED,
1	4	6	0	0	0	0	0	0	0	4	2	0	0	0	0	REMOTEUNIT	RemoteUnit		A remote unit can be a RTU, IED,
1	4	6	0	0	0	0	0	0	0	4	2	0	1	0	9	REMOTEUNIT_RTUCATALOG	rtuCatalog	RTUCATALOG_REMOTEUNIT	Catalog of remote terminal
1	4	6	0	0	0	0	0	0	0	4	2	0	2	1	9	REMOTEUNIT_REMOTEPOINT	remotePoint	REMOTEPOINT_REMOTEUNIT	For a RTU remote points
1	4	6	0	0	0	0	0	0	0	4	2	0	3	1	9	REMOTEUNIT_COMMUNICATIONLINK	communicationLink	COMMUNICATIONLINK_REM	The connection to remote units
1	4	6	0	0	0	0	0	0	0	4	2	0	4	0	9	REMOTEUNIT_SIGNAL	signal	SIGNAL_REMOTEUNIT	A signal models any
1	f	5	0	0	0	0	0	0	0	4	1	0	0	0	0	REMOTEPOINT	RemotePoint		For a RTU remote points
1	f	5	0	0	0	0	0	0	0	4	1	0	1	0	4	REMOTEPOINT_SCADAID	scadaID		ID in the SCADA system. It
1	f	5	0	0	0	0	0	0	0	4	1	0	2	0	3	REMOTEPOINT_VALUESOURCE	valueSource		MeasurementValueSource
1	f	5	0	0	0	0	0	0	0	4	1	0	3	0	9	REMOTEPOINT_REMOTEUNIT	remoteUnit	REMOTEUNIT_REMOTEPOINT	A remote unit can be a RTU, IED,
1	f	5	0	0	0	0	0	0	0	4	1	0	4	0	9	REMOTEPOINT_SIGNAL	signal	SIGNAL_REMOTEPOINT	A signal models any
1	f	6	0	0	0	0	0	0	0	4	3	0	0	0	0	RTUCATALOG	RtuCatalog		Catalog of remote terminal
1	f	6	0	0	0	0	0	0	0	4	3	0	1	0	a	RTUCATALOG_REMOTEUNITTYPE	remoteUnitType	RemoteUnitType	Type of remote unit.
1	f	6	0	0	0	0	0	0	0	4	3	0	2	0	5	RTUCATALOG_COSTPERUNIT	costPerUnit		
1	f	6	0	0	0	0	0	0	0	4	3	0	3	0	3	RTUCATALOG_ANALOGSIGNALCOUNT	analogSignalCount		
1	f	6	0	0	0	0	0	0	0	4	3	0	4	0	3	RTUCATALOG_ANALOGCOMMANDCOUNT	analogCommandCount		Number of analog
1	f	6	0	0	0	0	0	0	0	4	3	0	5	0	3	RTUCATALOG_DIGITALSIGNALCOUNT	digitalSignalCount		
1	f	6	0	0	0	0	0	0	0	4	3	0	6	0	3	RTUCATALOG_DIGITALCOMMANDCOUNT	digitalCommandCount		Number of digital
1	f	6	0	0	0	0	0	0	0	4	3	0	7	1	9	RTUCATALOG_REMOTEUNIT	remoteUnit	REMOTEUNIT_RTUCATALOG	A remote unit can be a RTU, IED,

Slika 4.2.2 Primer generisanog konfiguracionog fajla u .xls formatu

Prvih 16 kolona koje su označene različitim bojama predstavljaju niblove (eng. *nibble*) [11]. *Nibble* predstavlja polovinu bajta tj. 4 bita. Kako prvih 16 kolona predstavljaju 16 *nibble*-ova, tako svaki niz od 16 *nibble*-ova čini jednu 64-bitnu reč. Svaka 64-bitna reč opisuje jedan red u generisanom fajlu.

Kolona *Name* koristi se za *C#* enumeracije ili *DB(Data Base)* imena kolona.

Kolona *Class/Attrib name* opisuje imena klasa i njihovih atributa.

Kolona *Type Additional Info* se koristi kao dodatna informacija za enumeracije.

Kolona *Description* predstavlja kratak opis klase ili njenog atributa.

4.2.1 Nibbles (n0-n15)

Kao što je ranije navedeno u generisanom fajlu prvih 16 markiranih kolona predstavljaju *nibble*-vi koji su označeni od **n0**-**n15**. Ono što treba napomenuti jeste da su čitljivi i ljudima a ne samo računaru. U nastavku detaljno će biti opisani svaki od njih.

n0 - Predstavlja ***model type*** tj. tip modela za koji će fajl biti generisan. Svaki model ima svoje ime i kod koji ga opisuje. U ovom primeru generisan je fajl za ***Network model*** i on je kodiran sa 1, što se može videti u prikazu generisanog fajla. Imena modela zajedno sa kodovima su smešteni u konfiguracionom fajlu i o tome će kasnije biti reči.

n1 - n7 - Su *nibble*-ovi koji predstavljaju ***inheritance hierachy*** tj. deo koda koji opisuje položaj klase u hijerarhiji nasleđivanja. Kasnije će biti detaljno objašnjen način na koji se kodovi dodeljuju klasama u hijerarhiji.

n8 - n11 - Su *nibble*-ovi koji predstavljaju ***DMS type code*** tj. deo koda koji se odnosi na klase koje su listovi (eng. leaf) u hijerarhiji. Svaki ***DMS type code*** je jedinstven za svaku klasu, za svaki paket i model. Ukoliko postoji klasa sa istim imenom u dva modela, ***DMS type code*** ce biti različit za te dve klase iako se isto zovu.

n12 - n13 - Su *nibble*-ovi koji predstavljaju ***attribute sequence number*** tj. redne brojeve atributa za svaku klasu posebno.

n14 – Predstavlja *nibble* koji daje informaciju o tome da li je atribut *single value* ili *sequence of values*. Ukoliko je *single value* kodira se kao 0, u slučaju da je *sequence of values* kodira se kao 1.

n15- Predstavlja ***property type*** tj tip atributa. Svaki tip se kodira na način predstavljen u tabeli 4.1.1.1.

Kod	Tip atributa
0	*empty*
1	*bool*
2	*byte*
3	*int32*
4	*int64*
5	*float*
6	*double*
7	*string*
8	*datetime*
9	*cross-reference*
a(10)	*enum*
b(11)	*struct*
c(12)	*timespan*
d(13)	*complex*

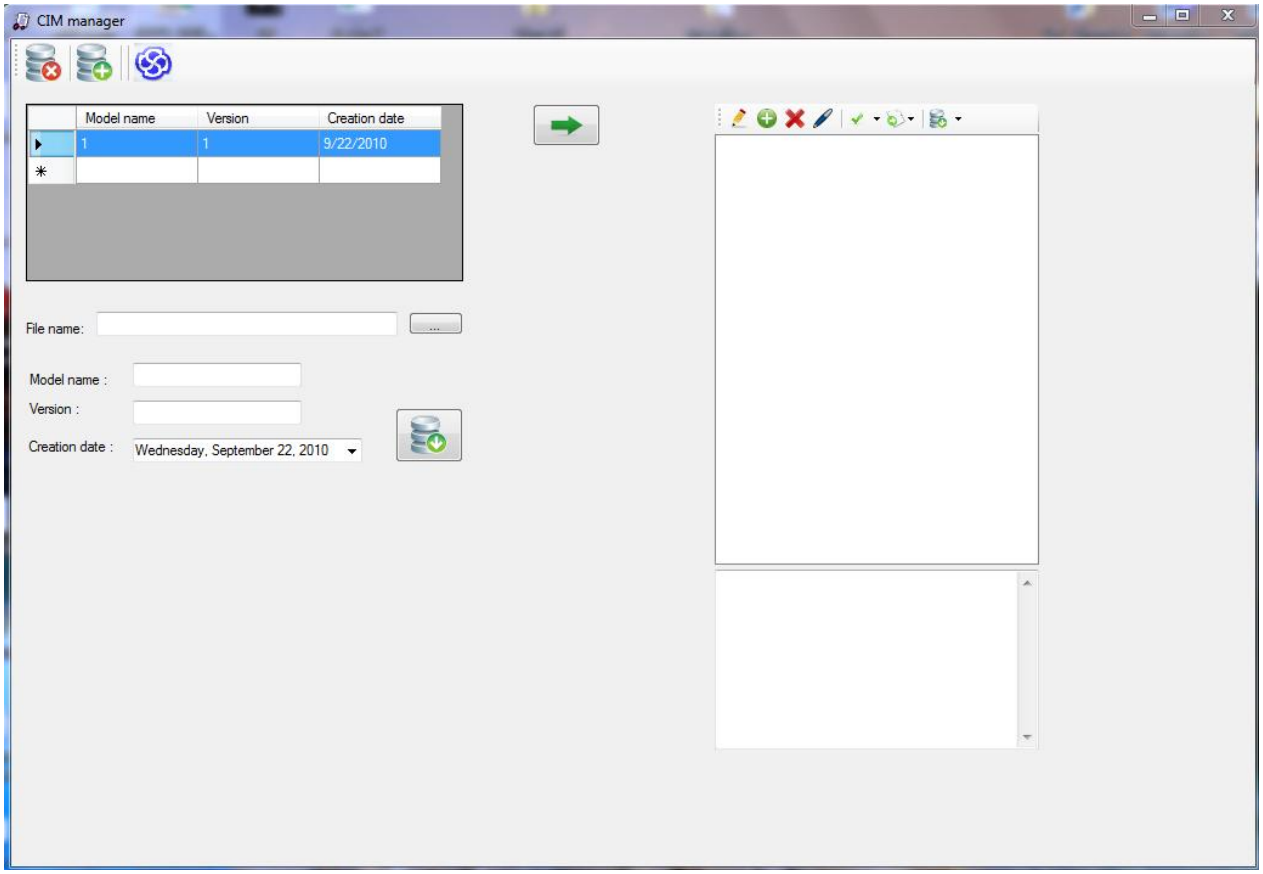
Tabela 4.2.1.1 Kodiranje tipova podataka

4.3 Aplikacija *CIM manager*

Aplikacija ***CIM manager*** predstavlja alat za čuvanje, manipulaciju, i generisanje *CIM* modela i *CIM* profila. Model se iz *EAP* fajla transformiše i čuva u relacionu bazu podataka (*Microsoft SQL*). Aplikacija je takođe realizovana u programskom jeziku *C#* kao *GUI* aplikacija pod *Visual Studio 2008* okruženjem.

Na slici 4.3.1 je prikazana glavna forma aplikacije.

Na samom vrhu može se primetiti glavna traka sa alatima (*eng. Tool Strip*)  . Prva ikonica se koristi za brisanje selektovanog modela iz baze. Druga ikonica predstavlja dodavanje kopije *CIM* modela (profila) u bazu podataka. Ta opcija je korisna ukoliko korisnik želi da kreira novi profil nad kojim može da radi izmene. Klikom na treću ikonicu omogućava se povezivanje izmenjenog *CIM* modela (profila) iz baze podataka i *CIM* modela (profila) koji je sačuvan u *EAP* fajlu. To je vrlo korisna opcija posebno ukoliko korisnik nema instaliran *Enterprise Arhitect* kod sebe, tako da ne zavisi od njega. Sve izmene koje obavi nad modelom u bazi podataka će se prilikom povezivanja sa *EAP* fajlom obaviti i u samom *EAP*-u.



Slika 4.3.1 Glavni prozor aplikacije ***CIM manager***

Edit polje ***File name*** predstavlja putanju do *EAP* fajla. Klikom na dugme  otvara se forma za izbor *EAP* fajla čijim izborom se popunjava dato polje.

Model name predstavlja ime modela koji će se sačuvati u relacionoj bazi pod tim imenom.

Version predstavlja verziju modela,dok je ***Creation date*** datum kreiranja EAP fajla.

Klikom na dugme  izabrani *EAP* model se čuva u relacionu bazu.

U svakom trenutku je moguće videti spisak modela (profila) koji se trenutno nalaze u bazi. To se može videti na glavnoj formi. Modeli su prikazani u tabeli i u ovom primeru se vidi ,da baza sadrži samo jedan model.

Da bi se lakše manipulisalo sa modelom (profilom) potrebna je njegova vizuelna reprezentacija. U svakom momentu model je moguće prikazati kao stablo (eng. *Tree View*) što se može videti na formi. Klikom na dugme  trenutno izabrani model se prikazuje kao stablo.

Iznad stabla se nalazi traka sa alatima (*eng. Tool Strip*) čija je osnovna namena manipulacija sa stablom. U nastavku je data tabela svih akcija koje mogu da se izvrše nad stablom.

	• Izmena klase • Izmena atributa • Izmena relacije (veze) između dve klase
	• Dodavanje nove klase • Dodavanje novog atributa klase
	• Brisanje klase • Brisanje atributa klase • Brisanje relacije(veze) između dve klase
	• Dodavanje relacije između dve klase
	• Označavanje svih elemenata stabla • Deselekcija svih elemenata stabla • Prikaz elemenata stabla koji su označeni
	• Export CIM modela(profila) u RDFS format
	• Čuvanje izmena u bazu

Tabela 4.3.1 Prikaz operacija koje mogu da se izvrše nad stablom

4.3.1 Manipulacija sa stablom

Kao što je već ranije rečeno, nad čvorovima stabla moguće je izvršiti određen skup operacija.

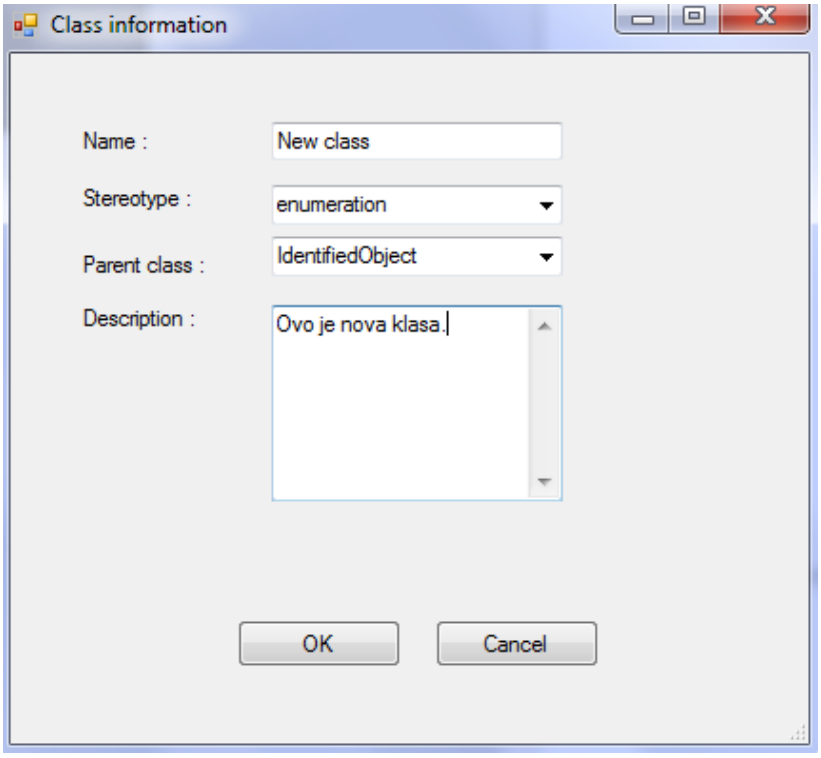
Paket je u stablu predstavljen sa:  , klasa sa:  , atribut:  , dok je veza predstavljena kao → ili ← u zavisnosti od smera orjentacije.

Ukoliko korisnik želi da doda novu klasu, potrebno je da selektuje paket u koji želi da doda klasu. Klikom na  otvara se forma prikazana na slici 4.3.1.1.

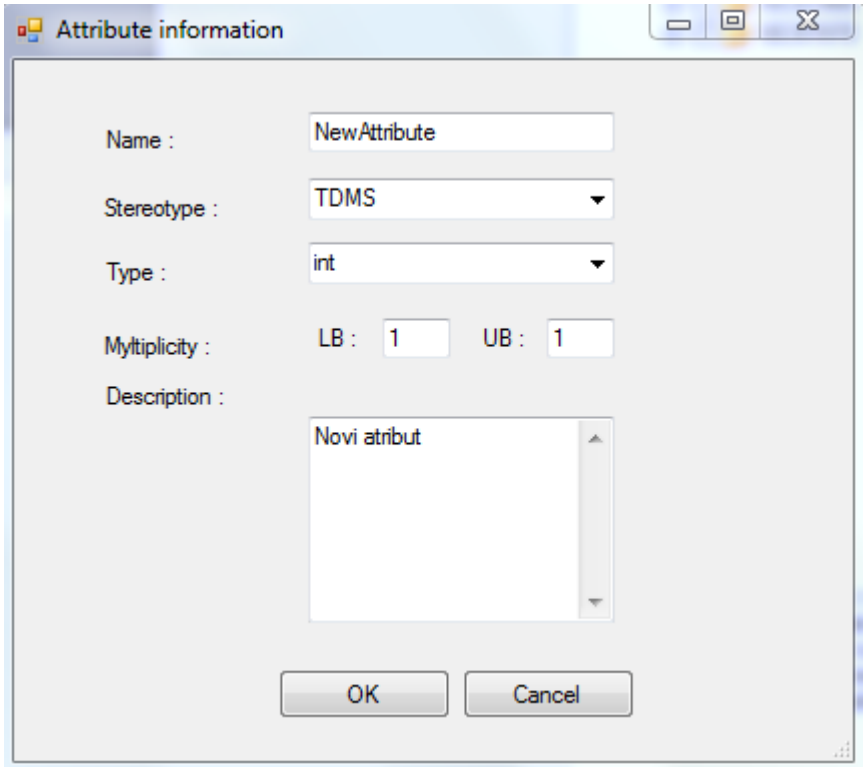
Potrebno je popuniti polja: ***Name*** (ime klase), ***Stereotype*** (stereotip), ***Parent* class** (klasa koju nasleđuje) i ***Description*** (opis koji karakterise datu klasu).

Ukoliko je korisnik prilikom unosa slučajno pogrešno uneo podatke koji opisuju datu klasu, u mogućnosti je da je obriše klikom na  , ili ukoliko želi da promeni podatke o njoj potrebno je da je selektuje u stablu i da klikom na  izvrši potrebne izmene.

Dodavanje atributa klase se može obaviti na sličan način s tim što korisnik prethodno mora da označi klasu kojoj želi da doda atribut i klikom na otvoriće se forma za dodavanje novog atributa.(Slika 4.3.1.2)



Slika 4.3.1.1 Dijalog za kreiranje nove klase



Slika 4.3.1.2 Dijalog za kreiranje novog atributa

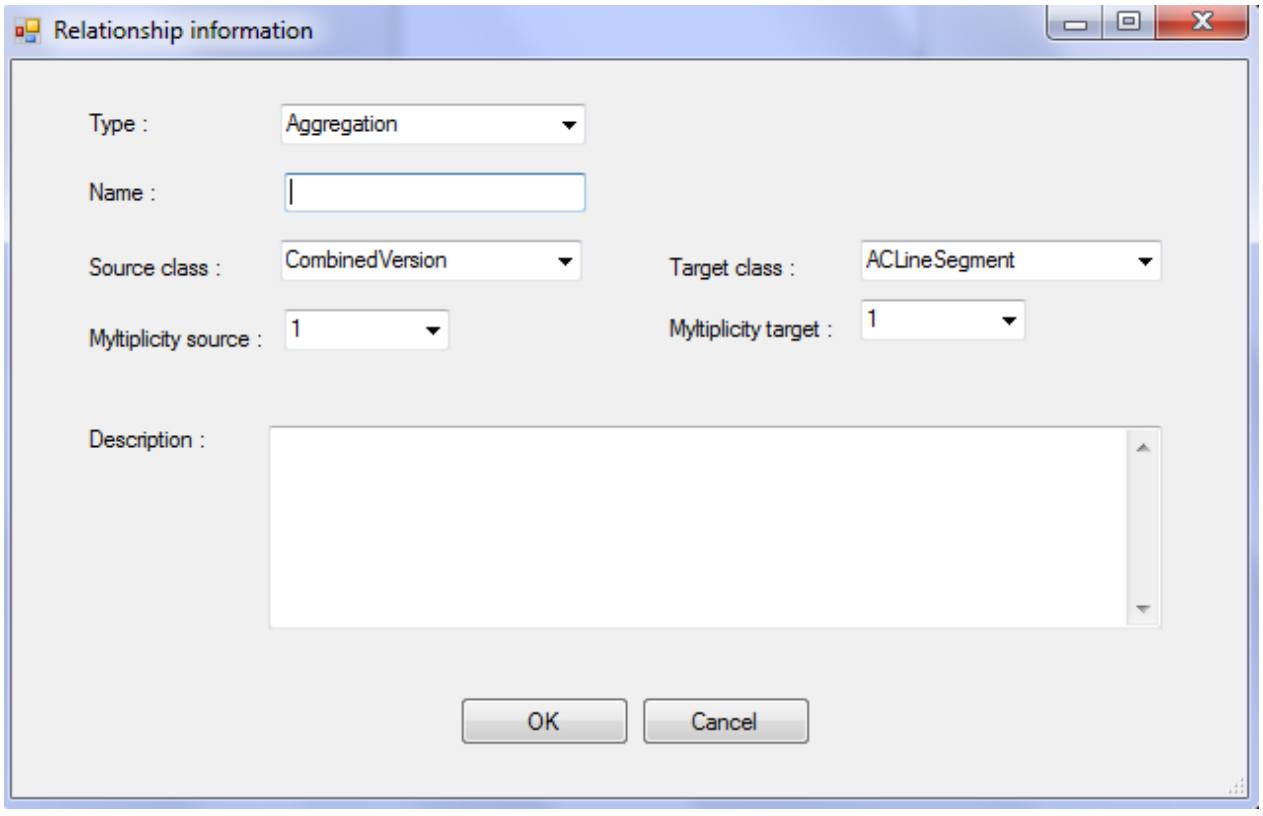
Potrebno je popuniti polja: ***Name*** (ime atributa), ***Stereotype*** (stereotip), ***Type*** (tip atributa) , ***Myltiplicity*** (donja i gornja granica) i ***Description*** (opis koji karakteriše datu klasu).

Ukoliko je korisnik prilikom unosa slučajno pogrešno uneo podatke koji opisuju dati atribut, u mogućnosti je da ga obriše klikom na , ili ukoliko želi da promeni podatke o njemu potrebno je da ga selektuje i da klikom na izvrši potrebne izmene.

Dodavanje relacije (veze) između dve klase je malo drugačije. Relaciju je moguće dodati klikom na pri čemu će se otvoriti dijalog prikazan na slici 4.3.1.3.

Potrebno je popuniti polja: ***Type*** (tip veze:agregacija,asocijacija), ***Name*** (ime veze), ***Source class*** (ime klase od koje veza počinje), ***Target class*** (ima klase na kojoj se veza završava), ***Myltiplicity source*** (kardinalitet ***Source*** klase), ***Myltiplicity target*** (kardinalitet ***Target*** klase) i ***Description*** (opis koji karakteriše datu klasu).

Ukoliko je korisnik prilikom unosa slučajno pogrešno uneo podatke koji opisuju datu relaciju, u mogućnosti je da je obriše klikom na , ili ukoliko želi da promeni podatke o njoj potrebno je da je selektuje i da klikom na izvrši potrebne izmene.



Slika 4.3.1.3 Dijalog za kreiranje nove relacije

4.3.2 Export *CIM* modela (profila) u *RDFS* fajl

U prethodnom podpoglavlju napomenuto je da je jedna od namena alata *CIM manager* i generisanje *CIM* profila. *CIM* profil se generiše klikom na iznad stabla. Izborom lokacije i naziva , generiše se *RDFS* fajl na osnovu čekiranih elemenata u stablu. Primer generisanog fajla je dat u listingu 4.3.2.1.

```
<?xml version="1.0" encoding="utf-8"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cims="http://iec.ch/TC57/1999/rdf-schema-extensions-19990926#"
xmlns:j0="http://iec.ch/TC57/2008/CIM-schema-cim13#"
xmlns:owl="http://www.w3.org/2002/07/owl#"
xmlns:dc="http://purl.org/dc/elements/1.1/"
xmlns:cim="http://iec.ch/TC57/2007/profile#"
xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
xmlns:basE="http://iec.ch/TC57/2007/profile">
      <rdf:Description rdf:about="http://iec.ch/TC57/2008/CIM-schema-
cim13#Package_Core">
            <cims:belongsToCategory rdf:resource="http://iec.ch/TC57/2008/CIM-
schema-cim13#Package_IEC61970"/>
            <rdfs:comment>Contains the core PowerSystemResource and
ConductingEquipment entities shared by all applications plus common collections
of those entities.</rdfs:comment>
            <rdfs:label>Core</rdfs:label>
            <rdf:type rdf:resource="http://iec.ch/TC57/1999/rdf-schema-
extensions-19990926#ClassCategory"/>
      </rdf:Description>
      <rdf:Description rdf:about="http://iec.ch/TC57/2008/CIM-schema-
cim13#BaseVoltage">
            <rdfs:subClassOf rdf:resource="http://iec.ch/TC57/2008/CIM-schema-
cim13#IdentifiedObject"/>
            <cims:belongsToCategory rdf:resource="http://iec.ch/TC57/2008/CIM-
schema-cim13#Package_Core"/>
            <rdfs:comment>Collection of BaseVoltages which is used to verify
that the BusbarSection rdfs:comment>
            <rdfs:label>BaseVoltage</rdfs:label>
            <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Class"/>
      </rdf:Description>
      <rdf:Description rdf:about="http://iec.ch/TC57/2008/CIM-schema-
cim13#BaseVoltage.nominalVoltage">
            <cims:stereotype
rdf:resource="http://langdale.com.au/2005/UML#attribute"/>
            <rdfs:comment>The PowerSystemResource's base
voltage.</rdfs:comment>
            <rdfs:label>nominalVoltage</rdfs:label>
            <cims:dataType rdf:resource="http://iec.ch/TC57/2008/CIM-schema-
cim13#Voltage"/>
            <rdfs:domain rdf:resource="http://iec.ch/TC57/2008/CIM-schema-
cim13#BaseVoltage"/>
            <cims:multiplicity rdf:resource="http://iec.ch/TC57/1999/rdf-
schema-extensions-19990926#M:0..1"/>
            <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Property"/>
      </rdf:Description>
      <rdf:Description rdf:about="http://iec.ch/TC57/2008/CIM-schema-
cim13#BaseVoltage.ConductingEquipment">
            <rdfs:comment>Use association to ConductingEquipment only when
there is no VoltageLevel container used.</rdfs:comment>
            <rdfs:label>ConductingEquipment</rdfs:label>
            <cims:inverseRoleName rdf:resource="http://iec.ch/TC57/2008/CIM-
schema-cim13#ConductingEquipment.BaseVoltage"/>
            <rdfs:range rdf:resource="http://iec.ch/TC57/2008/CIM-schema-
cim13#ConductingEquipment"/>
            <rdfs:domain rdf:resource="http://iec.ch/TC57/2008/CIM-schema-
cim13#BaseVoltage"/>
            <cims:multiplicity rdf:resource="http://iec.ch/TC57/1999/rdf-
schema-extensions-19990926#M:0..*"/>
```

```
            <rdf:type rdf:resource="http://www.w3.org/2000/01/rdf-
schema#Property"/>
</rdf:Description>
</rdf:RDF>
```

Listing 4.2.2.1 Primer generisanog *RDFS* dokumenta

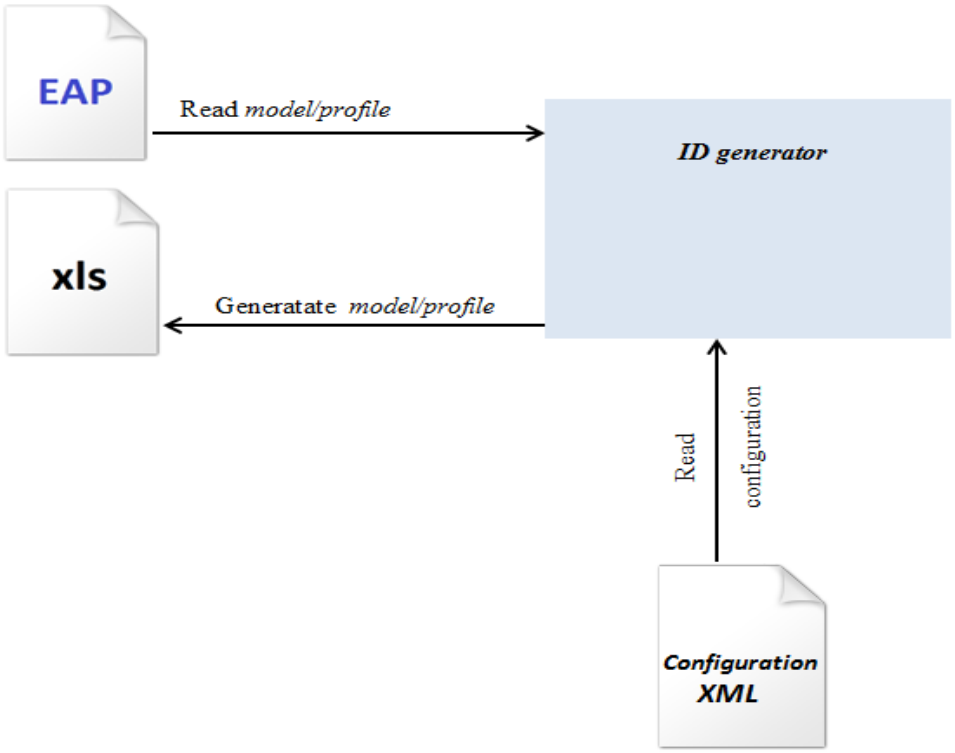
5. DETALJI IMPLEMENTACIJE SOFTVERSKOG ALATA

U prethodnom poglavlju su opisane aplikacije na način koji bi omogućio lakše razumevanje od strane običnog korisnika. U ovom poglavlju će biti više reči o samim detaljima implementacije.

5.1 Aplikacija *ID generator*

Na slici 5.1.1 je prikazana uprošćena arhitektura aplikacije ***ID generator***.

Aplikacija ***ID generator*** učitava CIM model/profil definisan u *EAP* fajlu. Na osnovu učitanog modela i konfiguracionog fajla (*Configuration XML*) kreira se objektni model (poglavlje 5.1.1). Iz objektnog modela se generiše *XLS* fajl koji predstavlja krajnji rezultat izvršenja aplikacije.



Slika 5.1.1 Uprošćena arhitektura aplikacije *ID generator*

U nastavku će biti više reči o samoj realizaciji nekih od funkcija koje su zadužene za transformaciju informacija dobijenih iz *EAP*-a i generisanje *XLS* fajla. Akcenat će biti stavljen na generisanje *nibble*-ova (poglavlja 5.1.2-5.15) koji predstavljaju jedinstveni kod za svaku klasu i za svaki generisani atribut iz fajla.

5.1.1 Učitavanje *CIM* modela definisanog u *Enterprise Architect*-u

Glavna klasa koja transformiše učitani model, definisan u *EAP* fajla, u objektni model jeste ***CModelReader***. Kada se dobije objektni model, lako je izgenerisati potreban *XLS*.

Objektni model je organizovan hijerarhijski. Postoji više klasa koje zajedno čine objektni model modela definisanog u *EAP* fajlu. ***CObjectModel*** je klasa koja sadrži listu paketa (***CPackage*** klasa) kao i ime modela. Klasa ***CPackage*** sadrži listu klasa (***CClass*** klasa), dok ***CClass*** sadrži listu atributa i relacija (***CAttribute*** klasa).

Samo učitavanje *EAP* modela je jednostavno. Potrebno je instancirati ***RepositoryClass*** klasu *Interop.EA API*-ja, u okviru ***CModelReader*** klase, i pozvati metodu *openFile* čiji je jedini argument putanja do *EAP* fajla, i model će biti učitan u memoriju. Kada je klasa instancirana moguće je pristupiti bilo kojem elementu modela programski. Ono što je sledeći korak jeste kreiranje objektnog modela (***CObjectModel***) na osnovu učitanog *EAP* fajla.

Pošto je *EAP* učitan, prolaskom kroz sve elemente fajla, kreira se objektni model koji reprezentuje klasa ***CObjectModel.*** Metoda *DoPackage* klase ***CModelReader*** omogućava prolazak kroz sve pakete učitanog fajla i kreiranje ***CPackage*** objekta za svaki paket iz *EAP*-a. Metoda *CollectClass* se poziva iz metode *DoPackage* i na osnovu nje se za svaku klasu iz *EAP*-a formira ***CClass*** objekat. Metoda *CollectAttributes* se poziva iz metode *CollectClass*, i za svaki atribut klase i vezu ka drugoj klasi iz *EAP*-a se formira ***CAttribute*** objekat.

5.1.2 Generisanje Model Type (n0)

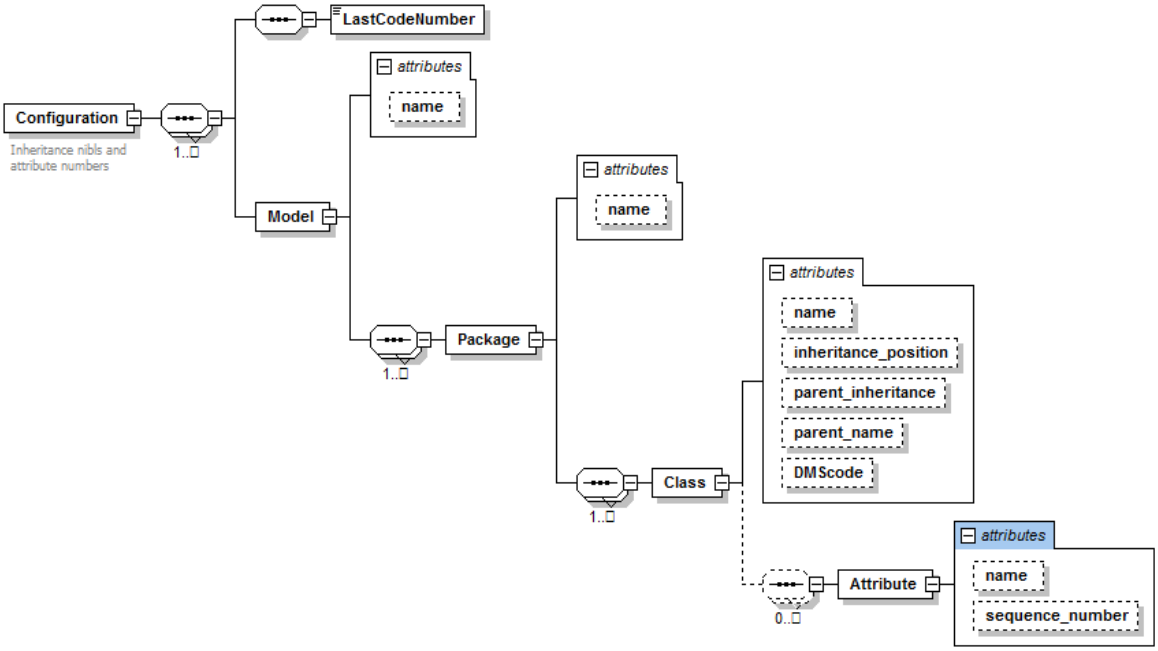
U zavisnosti od modela za koji treba da se generiše fajl, zavisi i vrednost *nibble* (*n0)*. Pošto je model podložan promenama, tako je moguće i da mu se promeni ime. Zbog toga su imena modela i kodovi smešteni u konfiguracionom fajlu *Modeltype.txt*. Svaki red fajla reprezentuje ime modela nakon čega se navodi njegov kod. Vrednosti kodova su proizvoljni, ali nakon dodele moraju ostati isti od verzije do verzije. Pošto je samo jedan nibble dodeljen kodovanju modela, to znači da je maksimalan broj modela koji može da se kodira 15.

5.1.3 Generisanje *inheritance hierachy (n1-n7)*

Samo generisanje n1-n7 nije previše komplikovano. Međutim, potrebno je jednom generisani kod održavati od verzije do verzije pod uslovom da klasa, za koju se generiše kod, ne promeni poziciju u hijerarhiji nasleđivanja. Ukoliko klasa promeni poziciju u hijerarhiji potrebno joj je pravilno dodeliti n1-n7 ali pod uslovom da ne naruši kodove koji su generisani za klase koje su i pre bile na tom hijerarhijskom nivou.

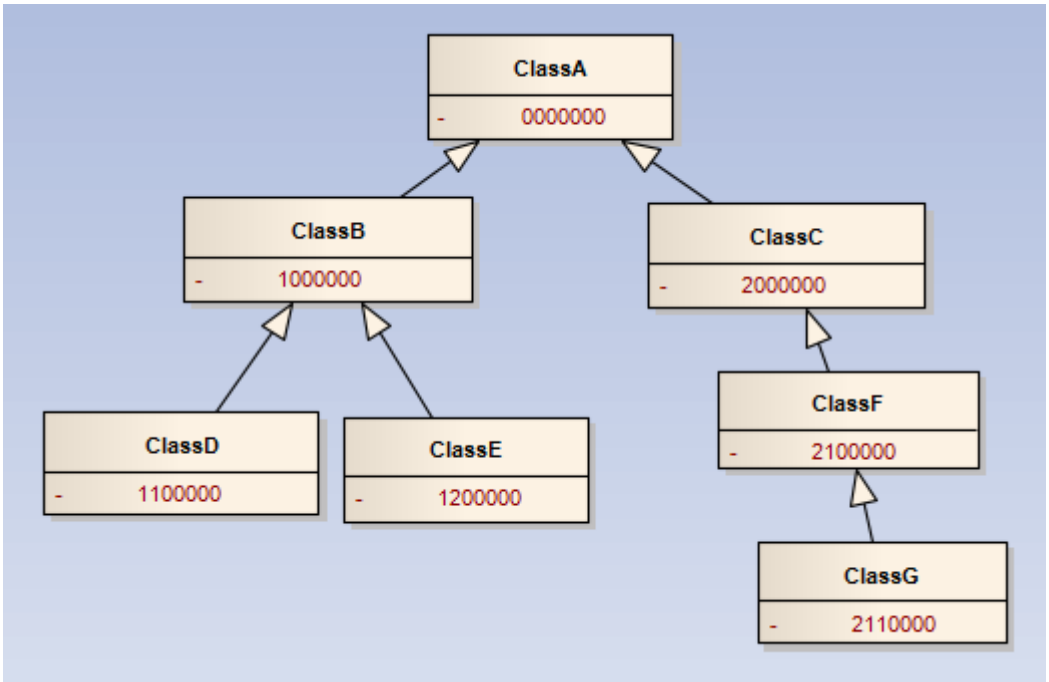
Da bi se podržala mogućnost čuvanja prethodnog stanja, tj hijerarhije nasleđivanja u prethodnoj verziji, u implementaciju je uveden konfiguracioni fajl *Configuration.xml*. Na slici 5.1.3.1 je data šema konfiguracionog fajla. Šema sadrži i druge informacije, koje će biti naknadno objašnjene u narednim poglavljima.

Korenski element generisanog *XML*-a je predstavljen kao **Configuration**. Konfiguracioni fajl može da sadrži više različitih modela(element **Model). Model** sadrži atribut **name** koji predstavlja ime modela za koji se generišu kodovi. Svaki **Model** sadrži sekvencu paketa (element **Package**). **Package** je opisan imenom paketa(atribut **name**) i sadrži sekvencu klasa(element **Class**). Atributi koji opisuju **Class** elemenat su ime klase(**name**), pozicija klase na istom hijerarhijskom nivou (**inheritance_position**), generisani nibble-ovi za njenu roditeljsku klasu(**parent_inheritance**), ime roditeljske klase(**parent_name**), kao i **DMScode** (4.2.1).



Slika 5.1.3.1 Šema konfiguracionog fajla *Configuration.xml*

CIM model je opisan tako da postoji samo jedna klasa koja se nalazi na vrhu hijerarhije. Svaka klasa može da nasledi jednu klasu, što znači da ne postoji višestruko nasleđivanje. Slika 5.1.3.2 ilustrativno pokazuje način na koji se generišu *nibble*-ovi n1-n7.

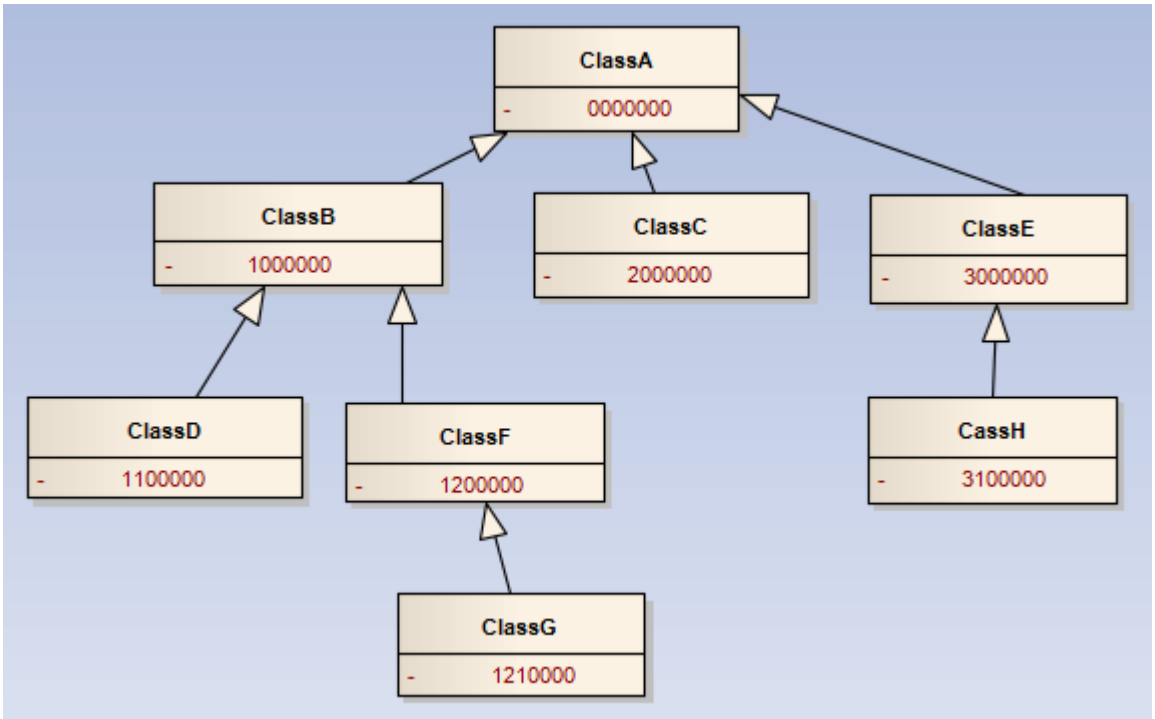


Slika 5.1.3.2 Način generisanja nibble-ova n1-n7

ClassA predstavlja klasu na vrhu hijerarhije i kao takva uvek se koduje sa 0000000. Klase koje se nalaze na drugom nivou hijerarhije se koduju tako sto se prvi *nibble* menja u zavisnosti od broja klasa koje se na tom nivou nalaze (prava klasa dobija vrednost n0=1, druga klasa n0=2,

petnaesta i svaka naredna n0=f). Na trećem nivou hijerarhije menjaju se prva dva *nibble*-a, na četvrtom prva tri (**ClassG**)…

Na slici 5.1.3.3 je prikazana situacija kada se hijerarhije u modelu menja, i kada se dodaje nova klasa u hijerarhiju (**ClassH**). Ukoliko se uporedi sa prethodnom slikom može se videti da su klase koje su zadržale istu poziciju u hijerarhiji zadržale i svoje kodove tj *nibble*-ove n1-n7. Klase koje su promenile hijerarhiju dobile su nove kodove (**ClassE, ClassF, ClassG**)



Slika 5.1.3.3 Način generisanja nibble-ova n1-n7 pri promeni hijerarhije

Do sada je bilo reči o situacijama o kojima treba voditi računa prilikom dodeljivanja kodova za nasleđivanje. U nastavku je dat pseudokod algoritma koji dodeljuje kodove klasama. Dodeljivanje kodova je realizovano kao rekurzija. Prilikom dodeljivanja kodova potrebno je funkciji proslediti parent klasu, koja će dobiti lista njenih naslednika. Za svakog naslednika, pozivaće se ista funkcija kojoj se prosleđuje konkretan naslednik koji ujedno predstavlja parenta za neke druge klase ukoliko postoje. Funkcija će se izvršavati dok god ima klasa u hijerarhiji.

```
setClassInheritance(classParent)
{
 childs = getChilds(classParent.name);
 foreach(child in childs)
  {
    Get generated inheritance Inher
    setChildInheritance(child.name,Inher);

    setClassInheritance(child);
   }
}
```

5.1.4 Generisanje *DMS Type code* (n8-n11)

DMS Type code je jednistven za svaku klasu. Jednom izgenerisan on treba da ostane isti dok god postoji i klasa za koju je generisan. Kodovi se čuvaju u *Configuration.xml* fajlu čija je šema već data na slici 5.1.3.1. Element **LastCodeNumber** predstavlja brojač *DMS* kodova, tj. čuva redni broj poslednjeg generisanog koda. Ta informacija je vrlo bitna, da bi se prilikom generisanja koda za neku novu klasu mogao izgenerisati naredni kod. Svaki **Class** element iz šeme opisan je atributom **DMScode** koji predstavlja kod date klase. Ukoliko klasa nije *leaf* u hijerarhiji, tada je njen kod 0000.

Prilikom generisanja koda za novu klasu, prvo se proverava da li je on već izgenerisan, proverom u konfiguracionom fajlu. Ukoliko se ne pronađe kod za tu klasu, generiše se novi na osnovu informacije o poslednjem generisanom kodu. Pošto je broj *nibble*-ova koji opisuju dati kod 4, maksimalan broj klasa tj. *leaf-ova* koji mogu dobiti kod je 65535.

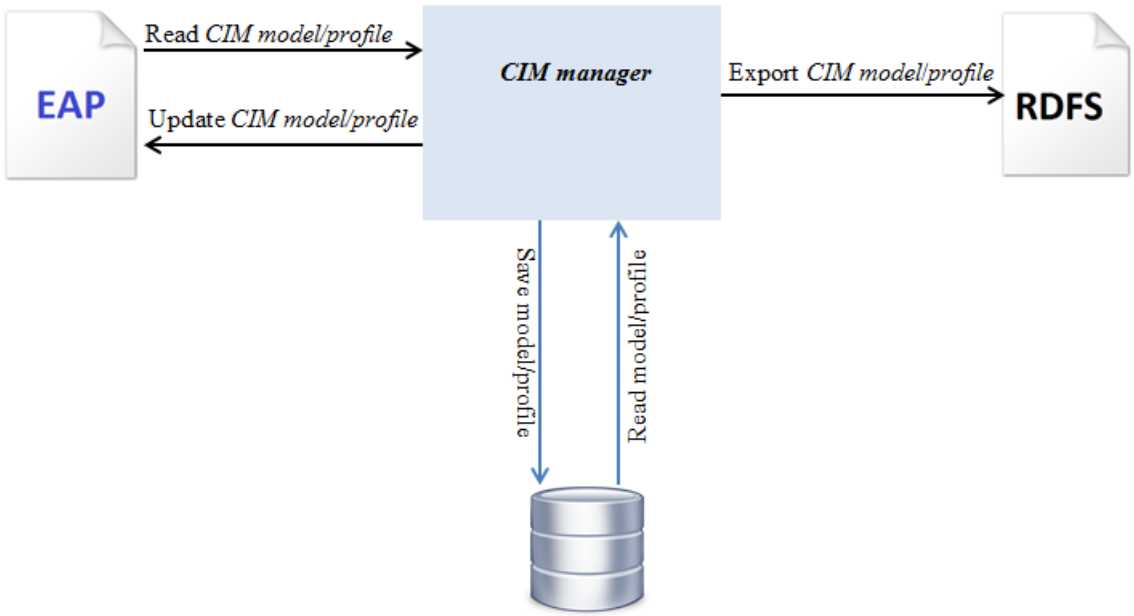
5.1.5 Generisanje *attribute sequence number* (n12-n13)

Ovaj deo koda predstavlja redni broj atributa klase. Takođe je bitno da jednom izgenerisani redni broj atributa ostane isti od verzije do verzije ukoliko i atribut ostane isti. Promena imena se evidentira kao brisanje atributa i dodavanje novog. Istorija se takođe čuva u konfiguracionom fajlu *Configuration.xml.* Elementom **Attribute** se opisuju atributi: **name** koji označava ime atributa, kao i **sequence_number** koji predstavlja redni broj atributa.(Slika 5.1.3.1)

Prilikom generisanja atributa treba voditi računa o tome da li se redni broj generiše po prvi put, i da li je redni broj već ranije izgenerisan. Ukoliko se redni broj generiše po prvi put njemu se dodeljuje prvi slobodan redni broj za datu klasu. Ukoliko već postoji, njemu se dodeljuje redni broj koji je već izgenerisan tj onaj redni broj koji se već nalazi u konfiguracionom fajlu.

5.2 Aplikacija *CIM manager*

U ovom poglavlju će biti opisana implementacija alata ***CIM manager***. Na slici 5.2.1 je prikazana uprošćena arhitektura aplikacije.



Slika 5.2.1 Uprošćena arhitektura aplikacije ***CIM manager***

Aplikacija ***CIM manager*** omogućava učitavanje (poglavlje 5.1.1) *CIM* modela/profila definisanog u *EAP* fajlu. Učitani fajl se transformiše u objektni model (poglavlje 5.2.2) koji se zatim čuva u bazi podataka (poglavlja: 5.2.1 i 5.2.3). Objektni model je moguće menjati (poglavlje

5.2.6) i sve izmene koje se obave nad modelom se mogu čuvati u bazi. Aplikacija omogućava čitanje modela/profila iz baze i njegov export u *RDFS* fajl (poglavlje 5.2.8). Jedna od korisnih funkcija jeste sinhronizacija objektnog modela iz baze i modela definisanog u *EAP* fajlu (poglavlje 5.2.7).

5.2.1 Model podataka *CIM manager* aplikacije

Kao što je ranije rečeno jedan od zadataka ovog rada jeste praćenje verzija *CIM* modela/profila. To je omogućeno uz pomoć baze podataka. Na slici 5.2.1.1 je dat konceptualni model šeme baze korišćene za praćenje verzija *CIM* modela/profila.

Model predstavlja *CIM* model ili profil koji je opisan imenom (***Model name***), njegovom verzijom (***Version***), i datumom kada je kreiran (***Creation date***). Pošto se u bazi čuvaju modeli i profili, ono što omogućava da se međusobno pravi razlika između njih jeste ime (***Model name***).

Svaki model može da sadrži više paketa (**Package**). Svaki paket ima svoj jedinstveni identifikator (***Package ID***), identifikator nadređenog paketa (**parentPackage**), identifikator koji ga jedinstveno opisuje u *Enterprise Arhitect*-u (***Package_EA_ID***), ime paketa (***Package name***), i opis (***Description***).

Svaki paket može sadržati više klasa (**Class**). Svaka klasu opisuje jedinstveni identifikator (***Class ID***), identifikator klase koju nasleđuje (***parentClass***), ime klase (***Name***), tip klase (***Type***), i njen opis (***Description***).

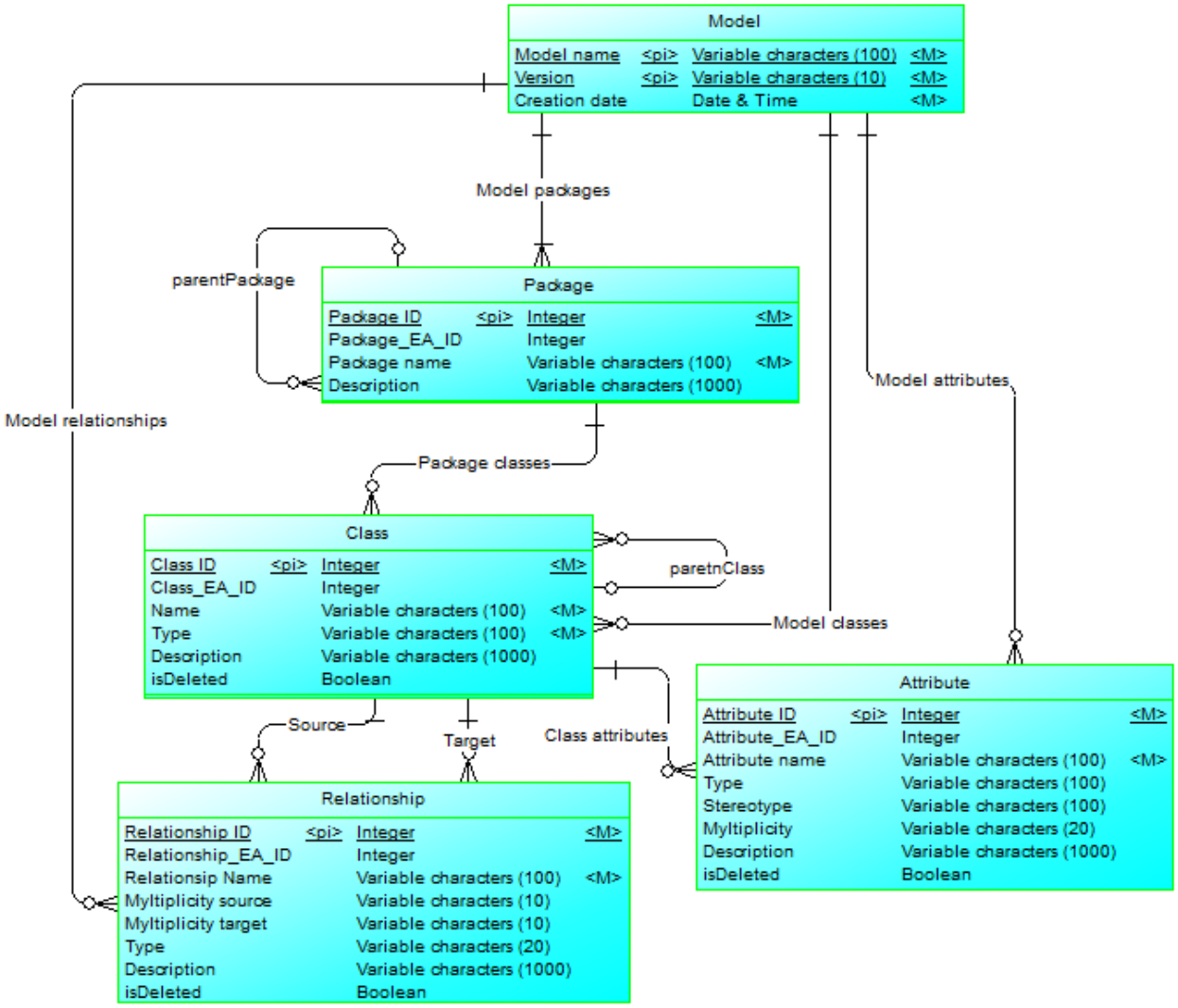
Svaka klasa može sadržati više atributa (**Attribute**) kao i veze (**Relationship**) sa drugim klasama.

Svaki atribut je opisan sa jedinstvenim identifikatorom (***Attribute ID***), identifikatorom koji ga jedinstveno opisuje u *Enterprise Arhitect*-u (***Attribute_EA_ID***), imenom (***Name***), tipom (***Type***), stereotipom (***Stereotype***), kardinalitetom (***Myltiplicity***), i opisom (***Description***).

Svaka relacija između klasa je opisana jedinstvenim identifikatorom (***Relationship ID***), identifikatorom koji je jedinstveno opisuje u *Enterprise Arhitect*-u (***Relationship_EA_ID***), imenom (***Name***), kardinalitetom klase od koje veza počinje (***Myltiplicity source***), kardinalitetom klase na kojoj se veza završava (***Myltiplicity target***), tip veze (***Type***), i opisom (***Description***).

Ono što se može primetiti jeste da je ***Model*** povezan sa svim ostalim entitetima (**Package**, **Class**, **Attribute**, **Relationship**). Ta veza je modelovana sa ciljem postizanja većih performansi upita, jer s vremenom se povećava broj modela i profila koji se smeštaju u bazu. Na osnovu imena modela i njegove verzije lako se mogu dobiti podaci iz ostalih tabela jer svaki red u tabeli sadrži informaciju kojem modelu pripada.

Atribut ***isDeleted*** u tabelama: **Class**, **Attribute** i **Relationship** predstavlja identifikator da je neka klasa, atribut ili relacija obrisana iz aplikacije. Ukoliko je vrednost atributa ***isDeleted*** **true**, znači da je atribut obrisan iz aplikacije. Ta informacija je potrebna prilikom povezivanja *CIM* modela iz baze i modela definisanog u *EAP* fajlu. O tome će više reči biti u poglavlju 5.2.7.



Slika 5.2.1.1 Model baze u koju se smeštaju *CIM* modeli/prfili

5.2.2 Objektni model *CIM* modela prethodno učitanog iz *EAP* fajla

Prilikom realizacije ove aplikacije nije korišćena *CObjectModel* klasa kao u poglavlju 5.1.1, ali je učitavanje samog modela ostalo isto. Ovde je korišćen objektni model koji je automatski izgenerisan na osnovu šeme baze .

Klasa ***ReadTransformCIM*** učitava *EAP* fajl, i na osnovu isčitanih elementa setuje vrednosti atributa objektnog modela generisanog od strane *Microsoft Visual Studio* okruženja. Potrebno je instancirati ***CIM_ModelDataContext*** klasu koja nasleđuje ***DataContext*** klasu.

DataContext klasa je izvor za sve entitete koji su mapirani preko konekcije sa bazom. Ona prati promene koje su napravljene nad svim preuzetim entitetima i održava "identite keša"(*identity cashe*) koji garantuje da su entiteti koji su preuzeti više od jednog puta predstavljeni istom instancom objekta. U principu ***DataContext*** instanca je dizajnirana da traje za samo jednu "jedinicu rada". ***DataContext*** je jednostavna i nije skupa za kreiranje. Tipična *LINQ to SQL* aplikacija kreira ***DataContext*** instance na nivou funkcije ili kao član klasa koje kratko žive, predstavljajući tako logički skup povezanih operacija baze podataka [14].

Klase: ***Model***, ***Package***, ***Class***, ***Attribute***, ***Relationship*** su automatski izgenerisane. Svaku je potrebno posebno instancirati u zavisnosti koliko *EAP* fajl sadrži paketa, koliko svaki paket

sadrži klasa, klasa atributa i relacija sa drugim klasama. Nakon instanciranja potrebno je setovati atribute kreiranih instanci.

5.2.3 Čuvanje učitanog modela

Kada je model učitan u memoriju i kada je formiran objektni model, vrlo je jednostavno sačuvati ga u bazu. Klasa koja je zadužena za čuvanje modela u bazu jeste ***StoreModel***. Konstruktoru ove klase je potrebno proslediti instancu ***CIM_ModelDataContext*** klase i instancu ***Model*** klase. Pozivom dve metode, *LINQ* objektni model čuva u bazu. To se može videti iz koda koji je dat u listingu 5.2.3.1.

```
public StoreModel(Model objectModel, CIM_ModelDataContext dataCont)
{
            dataContext.Models.InsertOnSubmit(objectModel);
            dataContext.SubmitChanges()
}
```

Listing 5.2.3.1 Konstruktor klase ***StoreModel***

Metoda *InsertOnSubmit* samo priprema entitet za unos u bazu podataka, stavljajući ga u listu, sve dok se ne pozove metoda *SubmitChanges()* koja će izvršiti izmene nad bazom.

5.2.4 Brisanje i čitanje modela(profila) iz baze

Sačuvani model(profil) je moguće obrisati ili učitati iz baze na zahtev korisnika. Klasa ***ReadOrRemove*** model poseduje dve metode: *getModel()* i *removeModel()*. Prva metoda čita model iz baze, dok ga druga briše. Kao argumenti i jedne i druge metode navode se ime i verzija modela. U nastavku su dati listinzi ove dve metode.

```
public Model getModel(String modelName,String modelVersion)
        {
            var queryModel = from model in dataContext.Models
                             where ((model.Model_name == modelName) &&
                                    (model.Version == modelVersion))
                             select model;
            foreach (var mod in queryModel)
            {
                return (Model)mod;
            }
            return null;

        }
```

Listing 5.2.4.1 Učitavanje modela iz baze

```
public void removeModel(String modelName,String modelVersion)
        {
            Model model = getModel(modelName, modelVersion);
            dataContext.Attributes.DeleteAllOnSubmit(model.Attributes);
            dataContext.Relationships.DeleteAllOnSubmit(model.Relationships);
            dataContext.Classes.DeleteAllOnSubmit(model.Classes);
            dataContext.Packages.DeleteAllOnSubmit(model.Packages);
            dataContext.Models.DeleteOnSubmit(model);
```

```
        //Promjena u bazi
        dataContext.SubmitChanges();
    }
```

Listing 5.2.4.2 Brisanje modela iz baze

5.2.5 Prikaz učitanog *CIM* modela (profila)

Da bi sa modelom(profilom) moglo nesmetano da se radi, potrebna je vizuelna reprezentacija onoga što se nalazi u bazi. Za prikaz je implementirana klasa ***CIMTreeView*** koja nasleđuje klasu ***TreeView*** koja predstavlja standardnu klasu *Windows Forms* aplikacija. ***TreeView*** prikazuje elemente kao hijerarhijsko stablo. Svaki element opisuje ***TreeNode*** klasa. Elementi koji su predstavljeni kao ***TreeNode*** su: paketi, klase, atributi i veze. Metoda *addModelToTree()* je zadužena za prikazivanje modela u vidu stabla. Kao argument metode navodi se instanca ***Model*** klase(opisana u poglavlju 5.2.2). U nastavku je dat pseudokod ove metode:

```
addModelToTree(Model model)
{
    List<TreeNode> packageNodeList = new List<TreeNode>();
    foreach (Package p in model.Packages)
    {
        Create new package node

        foreach (Class c in p.Classes)
        {
            Create new class node

            foreach (Attribute a in c.Attributes)
            {
              Create new attribute node
            }
            foreach (Relationship r in c.Relationships)
            {
                Create new relationship node
            }

            Add class node to package node
        }

        Add package to packageNodeList
    }
    makeTreeView(packageNodeList);
}
```

Listing 5.2.5.1 Pseudokod metode za prikaz modela

5.2.6 Operacije nad stablom (dodavanje, izmena i brisanje čvorova stabla)

Već je ranije bilo reči o operacijama koje mogu da se izvode nad stablom(poglavlje 4.3.1). Ovde će više reči biti o samoj implementaciji funkcija koje su zadužene za: dodavanje, izmenu i brisanje elemenata stabla. Treba napomenuti da izvršavanje ovih operacija direktno utiče na promenu objektnog modela čija ja vizelna reprezentacija stablo.

Dodavanje predstavlja jednu od operacija koja može da se obavi nad stablom. Moguće je dodati: klasu, atribut ili vezu (relaciju) između dve klase.

Prilikom dodavanja klase, instancira se objekat klase **Class**, čije se vrednosti promenljivih setuju vrednostima unesenim sa forme. Zatim se poziva funkcija *addClassNode* unesenim (klasa

CIMTreeView), koja novu klasu dodaje u stablo, kao i u objektni model. Prilikom dodavanja klase u stablo potrebno je funkciji proslediti trenutno selektovani čvor, koje ujedno predstavlja parenta za klasu, tj paket u koji će klasa biti dodata.

Prilikom dodavanja atributa, instancira se objekat klase **Attribute**, čije vrednosti promenljivih se setuju vrednostima unesenih sa forme. Poziv funkcije *addAttributeNode* (klasa ***CIMTreeView***), koja novi atribut dodaje u stablo i u objektni model. Funkcija *addAttributeNode* prima kao parametar instancu klase **Attribute** kao i trenutno selektovani čvor, koji predstavlja klasu u koju će se dodati atribut.

Prilikom dodavanja veze, instancira se objekat klase **Relationship**, čije se vrednosti promenljivih setuju vrednostima unesenih sa forme. Poziv funkcije *addRelationship* (klasa ***CIMTreeView***), obavlja dodavanje čvora, koji predstavlja vezu, u stablo kao i u objektni model. Čvor koji reprezentuje vezu u stablu se dodaje na dva mesta i to: kao dete čvora koji predstavlja *source* klasu, i kao dete čvora koji predstavlja *target* klasu.

Izmena predstavlja operaciju koju je moguće obaviti nad čvorovima stabla. Funkcije koje obavljaju izmenu čvorova su *updateClassNode*, *updateAttributeNode*, *updateRelationshipNode*. Kao parametri funkcija prosleđuju se objekti klase koji reprezentuju dati čvor, kao i selektovani čvor tj čvor nad kojim se obavljaju izmene. Kao rezultat ovih funkcija, vrše se izmene u stablu kao i izmene u objektnom modelu.

Brisanje je još jedna operacija koja je podržana u ovoj aplikaciji.

Brisanje atributa se obavlja funkcijom *removeAttributeNode*. Kao rezultat izvršenja funkcije briše se atribut iz stabla, a u objektnom modelu se označava kao obrisan, tj vrednost atributa *isDeleted* se postavlja na **true**(poglavlje 5.2.1).

Brisanje veze između dve klase se obavlja funkcijom *removeRelationshipNode*. Kao rezultat izvršenja funkcije, briše se veza između dve klase iz stabla, a u objektnom modelu se označava kao obrisan, tj. vrednost atributa *isDeleted* se postavlja na **true**.

Brisanje klase je nešto komplikovanije. Ukoliko korisnik obriše izabranu klasu iz stabla, potrebno je obrisati i svu decu čvora koji reprezentuje klasu (atribute i relacije). Metoda zadužena za brisanje čvora koji predstavlja klasu u stablu je *removeClassNode*. U nastavku je dat pseudokod ove metode (listing 5.2.6.1). Ukoliko se neka klasa obriše ona može da bude roditelj nekoj od klasa u objektnom modelu. Ukoliko se uspostavi da jeste, potrebno je anulirati sve atribute klasa iz modela koji predstavljaju ime parent klase. To obavlja metoda *removeParentFromClass* iz listinga.

```
removeClassNode(classNode,dataContext)
{
  List<TreeNode> relationshipForRemove = new List<TreeNode>();
  foreach (TreeNode child in classNode.Nodes)
  {
                If child is attribute
                {
                   Set attribute isDeleted=true;
                }
                else if relationship
                {
                  Set relationsip isDeleted=true;
                  Add relatinship for remove in relationshipForRemove list
                }
            }
            foreach (TreeNode node in relationshipForRemove)
                node.Remove();

            removeParentFromClass(classToRemove);
            Set class isDeleted = true;
            Remove class and attributes from treeView
   }
```

Listing 5.2.6.1 Pseudokod funkcije za brisanje klase iz stabla

Sve izmene koje se dešavaju na stablu neće prouzrokovati promene modela u bazi podataka. Izmene će biti validne samo ukoliko se obavi akcija za čuvanje izmena u bazi (Tabela 4.2.1). To će se obaviti pozivom jedne linije koda:

```
dataContext.SubmitChanges();//sve izmjene ce se desiti u bazi.
```

5.2.7 Povezivanje modela iz baze i modela sačuvanog u *EAP* fajlu

Klasa zadužena za povezivanje modela iz baze i modela definisanog u *EAP* fajlu jeste ***UpdateEAP***. Listing 5.2.7.1 predstavlja pseudokod metode *updateEAP* klase ***UpdateEAP*** koja obavlja proces sinhronizacije sa *EAP* modelom.

```
updateEAP(Model model)
{
   foreach (Class clas in model.Classes)
   {
    if (clas.isDeleted == true)
          Delete clas from EAP;
    else
    {
     try
       {
         Get classID from clas;
         If classID is in EAP update class from EAP;
       }
       catch (Exception e)
       {
         If classID is not in EAP;
         Create class and add to package in EAP;
       }

        foreach (Attribute attribute in clas.Attributes)
        {
         if (attribute.isDeleted==true)
               Delete attribute from EAP;
         else
         {
          try
          {
            Get attributeID from attribute;
            If attributeID is in EAP update attribute from EAP;
          }
           catch (Exception e)
           {
             If attributeID is not in EAP;
             Create new attribute and add to class;
           }
          }
         }
                Change eap with changed class;
       }
      }

            foreach (Relationship rel in model.Relationships)
            {
              if (rel.isDeleted==true)
               Delete rel from EAP;
              else
              {
```

```
                try
                {
                    If rel.ID is in EAP;
                     Update rel;
                }
                catch (Exception e)
                {
                  Get source class;
                  Get target class;
                  Create relationship and link source and target class;
                  Update eap with new relationship;
                }
              }
            }
          Close repository of eap file.
}
```

Listing 5.2.7.1 Pseudokod za sinhronizaciju sa *EAP* fajlom

Kao jedini parametar funkcije *updateEAP* prosleđuje se **Model** tj. objektni model koji će se mapirati na relacionu bazu (poglavlje 5.2.2). Prolaskom kroz klase modela dobija se *ID* klase koji se upoređuje sa *ID*-em u *EAP* fajlu. Ukoliko se pronađe klasa sa traženim *ID*-om znači da ona već postoji u njemu i rade se samo izmene *property*-ja klase koje su urađene u aplikaciji a nisu u *EAP* fajlu. Ukoliko se ne pronađe tražena klasa, to znači da ne postoji i da je treba dodati u *EAP* fajl. Slična je situacija i za atribute i relacije između klasa.

Ono što se takođe proverava jeste da li je vrednost atributa *isDeleted* **true**. Ako jeste znači da je data klasa, atribut ili relacija obrisana iz aplikacije, i da je treba obrisati i iz *EAP* fajla. Ovo predstavlja jednostavan način da se utvrdi koja je klasa, atribut ili relacija obrisana iz aplikacije. Ukoliko u bazi ne bi postojao atribut *isDeleted*, ne bi imali informaciju o tome koja klasa, atribut, ili relacija je obrisana. Jedini način da se to proveri jeste da se prolazi kroz sve elemente *EAP* fajla i da se na osnovu njihovog *ID* pronalaze odgovarajući elementi u modelu iz baze. Ukoliko se dati elemenat ne pronađe u modelu iz baze, to bi značilo da je on obrisan iz aplikacije. Međutim to je vremenski zahtevniji proces i ovo rešenje predstavlja jednostavniju varijantu. Nakon svakog povezivanja sa *EAP* fajlom, svaka klasa, atribut i relacija čija je vrednost *isDelete* atributa **true** se brišu iz baze podataka. Takođe svaki elemenat koji je dodat iz aplikacije, u bazi ne poseduje *ID* iz *EAP*-a. Prilikom povezivanja sa *EAP* fajlom, svaki takav element dobija svoj *EAP ID* i takve izmene se evidentiraju u bazi.

5.2.8 Export *CIM* objektnog modela u *RDFS* fajl

CIM model je u memoriji predstavljen kao objektni model i kao takav je pogodan za manipulaciju unutar same aplikacije. Lako se vizuelno može predstaviti kao stablo o čemu je ranije bilo reči. Za razmenu podataka preko mreže pogodno je prebaciti podatke u format pogodan za brz i jednostavan prenos podataka koji je ujedno čitljiv od strane ljudi i aplikacija. Zbog toga je potrebno izvršiti eksport podataka iz objektnog modela *CIM* modela u tekstualnu formu, odnosno u *RDF Schema (RDFS)* format.

Aplikacije sa internom ili eksternom podrškom za *CIM* mogu izvršiti učitavanje podataka iz *CIM XML* dokumenta. Zavisno od implementacije *CIM*-a u aplikacijama sa internom podrškom za *CIM* učitaće se samo podaci koji opisuju objekte klasa podržanih konkretnom implementacijom. Aplikacije koje imaju eksternu podršku za *CIM* pri učitavanju podataka moraju izvršiti konverziju podataka u svoj interni model.

Klasa koja je zadužena za eksport *CIM* objektnog modela u *RDFS* format je ***CreateRDFS***. Proces exporta je dat je pseudokodom opisanim u listingu 5.2.8.1.

```
List<Entity> entities = GetCheckedEntities();
foreach (Entity e in entities)
{
 WriteToFile(e.ToRDF());
 SEQUENCE attributes = GetCheckedAttributes(e);
 foreach (Attribute attr in attributes)
   WriteToFile(attr.ToRDF());
}
```

Listing 5.2.8.1 Pseudokod generisanja(eksporta) *CIM* objektnog modela u *RDFS* format

Za svaki element u *CIM*-u preuzima se skup njegovih atributa [15]. Po *RDF*-u, svaki element predstavlja resurs i opisan je atributima i njihovim vrednostima. Zato se svaki objekat u *CIM*-u eksportuje u poseban *XML* element, a njegovi atributi u njegove podelemente u *XML* dokumentu.

Po prethodnom algoritmu svi podaci koji su prethodno označeni (*checked*) će biti eksportovani u *RDFS* format. Označavanje se vrši na nivou atributa. Na taj način za potrebe određenih aplikacija može se izvršiti eksport samo neophodnih atributa za odabrane klase. Ovako dobijen *RDFS* dokument sadrži sve potrebne podatke uz minimalnu veličinu dokumenta čime se ubrzava prenos dokumenta preko mreže i učitavanje podataka od strane aplikacije.

6. PERFORMANSE REALIZOVANIH APLIKACIJA

Pored opisanih funkcionalnosti veoma važan aspekat predstavlja i vreme potrebno za izvršenje određenih operacija aplikacija. U nastavku su date tabele koje opisuju performanse obe aplikacije. Predstavljeni su samo rezultati za operacije koje iziskuju najviše vremena.

U tabeli 6.1 prikazano je vreme potrebno za generisanje konfiguracionog fajla, za određeni broj klasa i atributa, od strane ***ID generator*** aplikacije.Treba napomenuti da vrednosti: **Broj klasa** i **Broj atributa** predstavljaju vrednosti klasa i atributa iz modela, a ne broj generisanih klasa i atributa u konfiguracionom fajlu.

Broj klasa	**Broj atributa**	**Vreme izvršavanja izraženo u sekundama**
157	291	~66
125	281	~58
95	237	~43
62	187	~32
58	187	~31
36	37	~19

Tabela 6.1 Prikaz performansi ***ID generator*** aplikacije

U tabeli 6.2 prikazano je vreme potrebno za učitavanje (iz *EAP* fajla) i smeštanje *CIM* modela/profila u relacionu bazu podataka. Pri razmatranju se vodilo računa o broju klasa i atributa klasa definisanih u izvornom *EAP* fajlu.

Broj klasa	**Broj atributa**	**Vreme izvršavanja izraženo u sekundama**
497	1568	~84
157	291	~36
125	281	~25
95	237	~22
62	187	~20
58	187	~18
36	37	~15

Tabela 6.2 Prikaz performansi ***CIM manager*** aplikacije

7. ZAKLJUČAK

U radu su opisana dva softverska alata: ***ID generator*** i ***CIM manager*.** Opisane su tehnologije i alati korišćeni u realizaciji konkretnih aplikacija. Dati su detalji implementacije određenih funkcija oba softverska alata, s osvrtom na pojedine specifične situacije.

Implementacijom ***ID generator*** aplikacije omogućeno je generisanje konfiguracionih fajlova u *XLS* formatu čija je upotreba višestruka. Recimo, za generatore koda servisa koji *host*-uju modele, generatore konfiguracionih fajlova za servise (uglavnom u *XML* formatu), generatore relacionog modela sistema (za izveštaje / *SQL* upite nad sistemom), dokumentaciju, s tim da je vrlo bitno da to sve bude sinhronizovano. Jedna od prednosti jeste da je vreme potrebno za generisanje *XLS* fajla neuporedivo kraće od vremena koje bi običan čovek upotrebio za njegovo kreiranje.

Aplikacija ***CIM manager*** takođe ima višestruku upotrebu. Njenom realizacijom omogućeno je čuvanje *CIM* modela (profila) u relacionoj bazi podataka. Na osnovu imena modela/profila, verzije i datuma kreiranja, moguće je pratiti različite verzije modela/profila. Aplikacija podržava operacije: dodavanja klasa, atributa, i relacija između klasa u sam model kao i brisanje i izmenu već pomenutih elemenata i čuvanje svih izmena u bazi. Na osnovu izabranih elemenata modela iz baze moguć je *export CIM* profila u *RDFS* format, koji je kao takav pogodan za razmenu između aplikacija. Jedna od prednosti aplikacije je sinhronizacija *CIM* modela/profila iz baze i modela/profila definisanog u *EAP* fajlu. Ukoliko se ispostavi da određeni *CIM* model nije potrebno čuvati, može se nesmetano ukloniti iz baze podataka.

Dalji pravci razvoja aplikacija su višestruki. Pre svega što se tiče ***ID generator*** aplikacije moguće je generisanje konfiguracionih fajlova drugih formata. Recimo, kreiranje *XML* formata koji predstavlja standardni format za razmenu između aplikacija. Takođe, što se tiče ***CIM manager*** aplikacije, moguć je export *CIM* modela/profila i u druge formate podataka u zavisnosti od potreba krajnjih korisnika(.xsd, .eap, .xls).

Integracija ova dva softverska alata u jedan, kojim bi aplikacija ***ID generator*** koristila podatke iz baze, koju koristi i ***CIM manager***. Samim tim bi bile potrebne izmene na šemi baze, dodavanjem još potrebnih informacija koje su neophodne za generisanje *.xls* fajlova. Tako bi se dobila kompleksnija baza podataka koju bi bilo potrebno dodatno optimizovati za upite koji bi se postavljali, s obzirom da bi količina podataka u njoj s vremenom eksponencijalno rasla. Iako vreme izvršavanja aplikacija nije stavljeno u prvi plan, na osnovu rezultata prikazanih u prethodnom poglavlju može se reći da i jedna i druga aplikacija pokazuju vrlo dobre rezultate što se tiče performansi. Međutim, uvek postoji težnja za boljim performansama tako da je još jedan pravac razvoja unapređenje brzine izvršavanja i jedne i druge aplikacije.

8. LITERATURA

[1] The Common Information Model for Distribution , An Introduction to the CIM for Integrating Distribution Applications and Systems, EPRI Project Manager, L. King, Technical Update, November 2008

[2] Draft IEC 61970: Energy Management System Application Program Interface (EMS-API) -Part 301: Common Information Model (CIM) Base, Second Edition

[3] *Unified Modeling Language (UML), version 2.1.2*: http://www.omg.org/technology/documents/formal/uml.htm

[4] ISO/IEC 19501:2005 Information technology — Open Distributed Processing — Unified Modeling Language (UML) Version 1.4.2 , http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=32620

[5] http://www.w3.org/

[6] http://en.wikipedia.org/wiki/XML **,** *Extensible Markup Language (XML)*

[7] http://www.w3.org/TR/2004/REC-owl-features-20040210/#s1, *Web Ontology Language (OWL)*

[8] Wrox, Microsoft Visual C# 2008, third 3rd Edition, Karli Watson, Christian Nagel, Jacob Hammer Pedersen, Jon D. Reid, Morgan Skinner, Eric White

[9] http://msdn.microsoft.com/en-us/library/bb386976.aspx, *LINQ to SQL*

[10] http://msdn.microsoft.com/en-us/library/Bb384429(v=VS.100).aspx, *Object Relational Designer*

[11] http://en.wikipedia.org/wiki/Nibble, *Nibble*

[12] http://www.codeproject.com/KB/linq/UnderstandingLINQ.aspx, picture of *LINQ* Arhitecture

[13] http://msdn.microsoft.com/en-us/library/bb399400.aspx, Code Generation in *LINQ to SQL*

[14] http://msdn.microsoft.com/en-us/library/system.data.linq.datacontext.aspx, *DataContext Class*

[15] Brankica Milosavljević. Implementacija konverzije podataka elektroenergetskog sistema iz jednog objektnog modela u objedinjeni informacioni model(Common Information Model), Fakultet Tehničkih Nauka, Novi Sad, 2008

[16]Wrox, Professional C#, Simon Robinson, Cristian Nagel, Karli Watson, Jay Glynn, Morgan Skinner, Bill Evjen

[17]Wrox, Proffesional Visual Studio 2008, Nick randolph, David Gardner

[18]Apress,Pro C# 2008 and the .NET 3.5 Platform,Fourth edition, Andrew Troeisen

[19] IEC 61970-501 Ed.1: Energy management system application program interface (EMS-API) - Part 501: Common information model resource description framework (CIM RDF) Schema