УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Саша Девић

Приступи развоју базе података Општег информационог модела за електроенергетске мреже

ДОКТОРСКА ДИСЕРТАЦИЈА

Нови Сад, 2018

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

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:	
Идентификациони број, **ИБР**:	
Тип документације, **ТД**:	Монографска документација
Тип записа, **ТЗ**:	Текстуални штампани материјал
Врста рада, **ВР**:	Докторска дисертација
Аутор, **АУ**:	Саша Девић
Ментор, **МН**:	др Иван Луковић, редовни професор
Наслов рада, **НР**:	Приступи развоју базе података Општег информационог модела за електроенергетске мреже
Језик публикације, **ЈП**:	српски
Језик извода, **ЈИ**:	српски
Земља публиковања, **ЗП**:	Србија
Уже географско подручје, **УГП**:	Војводина
Година, **ГО**:	2018
Издавач, **ИЗ**:	Факултет техничких наука
Место и адреса, **МА**:	Трг Доситеја Обрадовића 6, 21000 Нови Сад
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)	7/110/62/6/10/6/6
Научна област, **НО**:	Електротехничко и рачунарско инжењерство
Научна дисциплина, **НД**:	Примењене рачунарске науке и информатика, Информациони системи
Предметна одредница/Кључне речи, **ПО**:	Општи информациони модел (CIM), моделовање базе података, генерисање кода, електроенергетске мреже, модел-код трансформације
УДК	
Чува се, **ЧУ**:	Библиотека Факултета техничких наука, Трг Доситеја Обрадовића 6, 21000 Нови Сад
Важна напомена, **ВН**:	
Извод, **ИЗ**:	Општи информациони модел (CIM) користи се за опис електроенергетске мреже и за размену података између оператера преносних електроенергетских система. Како је модел постајао све заступљенији, појавила се потреба за његовим складиштењем. У раду је развијен методолошки приступ за развој базе података која би подржала релативно једноставно складиштење и рад са инстанцама CIM модела, које описују тренутно, активно стање у систему. Такође, омогућено је и праћење претходних, историјских стања CIM инстанци, као и њихова рестаурација у жељено стање. Очекује се да предложени приступ олакша увођење CIM модела у различита, наменска програмска решења.
Датум прихватања теме, **ДП**:	
Датум одбране, **ДО**:	

Чланови комисије, **КО**:	Председник:	Др Горан Швенда, редовни професор	
	Члан:	Др Мило Томашевић, редовни професор	
	Члан:	Др Славица Кордић, доцент	Потпис ментора
	Члан:	Др Дину Драган, доцент	
	Члан, ментор:	др Иван Луковић, редовни професор	

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:	
Identification number, **INO**:	
Document type, **DT**:	Monograph documentation
Type of record, **TR**:	Textual printed material
Contents code, **CC**:	Ph.D. thesis
Author, **AU**:	Saša Dević
Mentor, **MN**:	Ivan Luković, Ph.D., Full Professor
Title, **TI**:	Approaches for Developing a Database for Common Information Model of Power Grids
Language of text, **LT**:	Serbian
Language of abstract, **LA**:	Serbian
Country of publication, **CP**:	Serbia
Locality of publication, **LP**:	Vojvodina
Publication year, **PY**:	2018
Publisher, **PB**:	Faculty of Technical Sciences
Publication place, **PP**:	Trg Dositeja Obradovića 6, 21000 Novi Sad
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)	7/110/62/6/10/6/6
Scientific field, **SF**:	Electrical Engineering and Computing
Scientific discipline, **SD**:	Applied Computer Science and Informatics, Information Systems
Subject/Key words, **S/KW**:	Common Information Model (CIM); database modeling; code generators; power grid; model to code transformation
UC	
Holding data, **HD**:	Library of Faculty of Technical Sciences, Trg Dositeja Obradovića 6, 21000 Novi Sad
Note, **N**:	
Abstract, **AB**:	Common Information Model (CIM) is used for describing power grid networks and data exchange among transmission system operators (TSO). As the model became widely used, there was a need to store such model. In this thesis we present a methodological approach to development of a database that supports relatively easy storing and managing CIM instances, which describe current, active state of the system. Also, tracking changes and restoring CIM instances to its previous states are supported. We expect that such methodological approach would ease the implementation of CIM model in various, domain specific software solutions.
Accepted by the Scientific Board on, **ASB**:	
Defended on, **DE**:	

Defended Board, **DB**:	President:	Goran Švenda, Ph.D., Full Professor	
	Member:	Milo Tomašević, Ph.D., Full Professor	
	Member:	Slavica Kordić, Ph.D., Assistant Professor	Menthor's sign
	Member:	Dinu Dragan, Ph.D., Assistant Professor	
	Member, Mentor:	Ivan Luković, Ph.D., Full Professor	

Резиме

Развој модела за опис електроенергетских мрежа, заснован на Општем информационом моделу (енгл. Common Information Model – CIM), обухвата поступке развоја архитектуре, као и обезбеђења операбилности и размене података између различитих оператера преносних електроенергетских система. CIM је модел који је од своје прве појаве имао доста измена. Мењање структуре података је вршено у циљу отклањања пропуста у моделу, бољег раздвајања логичких целина, али и достизања структуре података која би била прихватљива за већину оператера преносних система. Поред тога, CIM је прихваћен од стране Европске мреже оператера преносних система за електричну енергију (енгл. European Network of Transmission System Operators for Electricity – ENTSO-E) као препоручени модел за опис преносних мрежа. Можемо сматрати да су то неки од разлога зашто је CIM, у релативно кратком временском року, постао широко прихваћен међу оператерима преносних система.

Како је модел постајао све заступљенији и како се наставило са развојем нових верзија модела, појавила се и потреба за складиштењем таквог модела. То доводи и до потребе да се креира методолошки приступ за развој базе података, која би подржала релативно једноставно складиштење и рад са инстанцама CIM модела које описују тренутно, активно стање у систему. Такође, неопходно је омогућити и праћење претходних, историјских стања CIM инстанци, као и њихову рестаурацију у жељено стање. У циљу обезбеђења имплементације приступа који је предложен у овом истраживању, у докторској дисертацији предложена је примена парадигме аутоматског програмирања. При томе, развијен је генератор програмског кода који олакшава развој шеме базе података и програмског кода за рад са базом података. На тај начин, обезбеђено је брже реаговање на промене у самом CIM моделу. Из тог разлога, можемо очекивати да развој и примена новог методолошког приступа олакшава увођење CIM модела како у постојећа, тако и у нова програмска решења.

кључне речи: Општи информациони модел; моделовање базе података; генератори кода; електроенергетска мрежа; модел-у-код трансформације

Summary

The ongoing development of model for power grid networks based on the Common Information Model (CIM), is dealing with design, operability and data exchange among various power grid operators. CIM model had many versions since his first release. Changes in the model structure were made in order to remove errors in design, create better segregation of classes, but also to create a data model that is acceptable by most power grid operators. CIM is accepted by European Network of Transmission System Operators for Electricity (ENTSO-E) as a default model for describing power grids. Because of that, we can conclude that those were some of the main reasons why CIM, in relatively short period, became widely accepted among transmission system operators (TSO).

As the model became more used and continued to be developed, there was a need to store such model. This paper presents a methodological approach to development of a database that supports storing and managing of current, active CIM instances. Also, tracking of past, historical values of CIM instances is supported, as well as possibility to restore CIM instance to its previous state. In order to facilitate the implementation of the proposed approach, we apply a paradigm of automatic programming. A code generator is developed to ease the work on creating database schema and API communication layer to facilitate the database capabilities. The new approach would also allow faster response on changes in CIM model itself. By this, we could expect that such methodological approach would contribute to a wider acceptance of CIM in the power grid networks.

keywords: Common Information Model (CIM); database modeling; code generators; power grid; model to code transformation

мами, тати и брату
јер су увек веровали у мене

Предговор

Иако је CIM одавно препознат као стандард у домену електроенергетских система, и даље постоје одређене потешкоће када је у питању његова примена у софтверским решењима. Да би CIM модел био коришћен у неком пројекту, потребно је добро познавање структуре самог модела, као и доменско знање из области примене. Како CIM настоји да обухвати опис свих делова једног електроенергетског система са великим бројем повезаних класа, он се може означити као комплексан модел. Због свог обима и комплексности, једно од места где постоје потешкоће у примени CIM модела је развој базе података за смештање CIM модела. Из истог разлога настају потешкоће када је потребно пратити измене и рестаурирати инстанце модела у пређашње стање. Једна од могућности да се превазиђу поменуте потешкоће јесте парадигма развоја заснованог на моделу. Главни предмет ове докторске дисертације је управо спајање у један методолошки приступ познавање структуре CIM модела и, пратећи парадигме аутоматског програмирања и развоја заснованог на моделу, развој базе података за управљање инстанцама CIM модела.

Ова докторска дисертација је настала током докторских академских студија студијског програма Енергетика, електроника и телекомуникације на Факултету техничких наука Универзитета у Новом Саду и ангажмана њеног аутора у настави на Катедри за електроенергетику и електроенергетски софтверски инжењеринг, као и током рада у једној од водећих компанија за развој софтвера за управљање електроенергетским системима. Основне идеје, правци и зачеци истраживања описаног у овој докторској дисертацији представљени су у оквиру елабората за квалификациони испит и пријаве теме за израду докторске дисертације током докторских студија. Описано истраживање ослања се на научне студије спроведене у оквиру пројекта „Интелигентни системи за развој софтверских производа и подршку пословања засновани на моделима”, који је под евиденционом ознаком ИИИ-44010, финансиран од стране Министарства просвете, науке и технолошког развоја Републике Србије, те и ова докторска дисертација може бити означена као један од исхода поменутог пројекта.

Аутор одаје посебну захвалност свом ментору, проф. др Ивану Луковићу за бескрајно стрпљење, разумевање и немерљив допринос током представљеног истраживања и писања ове докторске дисертације. Такође, аутор се захваљује и члановима комисија које су пратиле, процењивале и усмеравале описано истраживање. Коначно, али ништа мање важно, аутор препознаје и цени доприносе бројних колега, сарадника и истраживача из разних институција и из различитих држава, који су, посредно или непосредно, омогућили долазак до научних резултата од значаја за истраживање у овој докторској дисертацији.

Садржај

Резиме 7

Summary 9

Предговор 13

Садржај 15

Списак слика 17

Списак табела 19

Списак графика 21

Списак прилога 23

Списак уведених појмова 25

Списак скраћеница 27

1. УВОД 31

1.1. Предмет и проблем истраживања 33

1.2. Циљ истраживања и очекивани резултати 34

1.3. Методе истраживања 36

1.4. Задаци истраживања 36

1.5. Хипотезе истраживања 37

2. ПРЕГЛЕД СТАЊА У ОБЛАСТИ 41

2.1. Стандардизација CIM модела 41

2.2. Практична примена CIM модела 42

2.3. Смештање CIM модела у базу података 44

2.4. Примена парадигме аутоматског програмирања 45

2.5. Алати за аутоматизацију развоја програмских апликација 51

2.6. Закључак поглавља 54

3. ПРОБЛЕМСКИ ДОМЕН И АНАЛИЗА МОГУЋИХ НАЧИНА МЕМОРИСАЊА ИНСТАНЦИ CIM МОДЕЛА 55

3.1. Основни појмови 55

3.2. Проблемски домен 58

3.3. Могући начини меморисања инстанци CIM модела 59

3.4. Закључак поглавља 62

4. ПРИСТУП РАЗВОЈУ АПЛИКАТИВНОГ РЕШЕЊА ЗА УПРАВЉАЊЕ ИНСТАНЦАМА CIM МОДЕЛА 63

4.1. Структура методолошког приступа 63

4.2. Ф1: Развој шеме базе података за смештање инстанци CIM модела и управљање инстанцама CIM модела 64

4.2.1. Ф1-К1: Анализа активног модела и његово поређење са CIM моделом 65

4.2.2. Ф1-К2: Дефинисање структуре активног модела ... *65*

4.2.3. Закључак фазе ... *66*

4.3. Ф2: Развој шеме базе података за праћење историје инстанци CIM модела ... 66

4.3.1. Ф2-К1: Дефинисање структуре за праћење промена над једним елементом ... *67*

4.3.2. Ф2-К2: Дефинисање структуре за праћење промена над групом елемената ... *69*

4.3.3. Ф2-К3: Дефинисање структуре за рестаурацију инстанце модела ... *70*

4.3.4. Закључак фазе ... *73*

4.4. Ф3: Развој поступка за рестаурацију инстанци CIM модела ... 73

4.4.1. Ф3-К1: Дефинисање алгоритма за поништавање појединачне измене ... *74*

4.4.2. Закључак фазе ... *77*

4.5. Закључак поглавља ... 77

5. РАЗВОЈ СОФТВЕРСКЕ АПЛИКАЦИЈЕ ЗА МЕМОРИСАЊЕ ИНСТАНЦИ CIM МОДЕЛА И УПРАВЉАЊЕ ИНСТАНЦАМА CIM МОДЕЛА ... 79

5.1. Пројектовање и реализација Генератора кода за потребе развоја циљне софтверске апликације ... 80

5.2. Пројектовање и реализација шеме базе података за подршку циљне софтверске апликације ... 82

5.3. Реализација поступака за смештање инстанци CIM модела у реализовану базу података и уградња у циљну софтверску апликацију ... 83

5.4. Реализација поступака за управљање инстанцама CIM модела и уградња у циљну софтверску апликацију ... 84

5.5. Реализација поступка за рестаурацију инстанци CIM модела и уградња у циљну софтверску апликацију ... 85

5.6. Закључак поглавља ... 86

6. АНАЛИЗА И ОЦЕНА РАЗВИЈЕНОГ ПРИСТУПА И СОФТВЕРСКОГ РЕШЕЊА ... 87

7. ЗАКЉУЧАК ... 93

Литература ... 97

Прилози ... 103

Списак слика

Слика 1: CIM UML дијаграм класа, исечак из EQ профила 56
Слика 2: Исечак шеме активног модела 64
Слика 3: Исечак шеме историјског модела 68
Слика 4: Историјски елементи 69
Слика 5: Групе разлика 70
Слика 6: Претходне разлике 72
Слика 7: Наредне разлике 73
Слика 8: Пресек разлика 74
Слика 9: Дијаграм тока алгоритма за поништавање појединачне измене 75
Слика 10: Развојни процес 79

Списак табела

Табела 1: Пример скупа података, табеле 58
Табела 2: Алгоритам за поништавање појединачне измене 76
Табела 3: Читање инстанце CIM модела из базе података 88
Табела 4: Упис инстанце CIM модела у базу података 89
Табела 5: Измена инстанце CIM модела са праћењем историје 90
Табела 6: Читање инстанце CIM модела из базе података и његова рестаурација 91

Списак графика

График 1: Писање CIM инстанци у базу података 60
График 2: Читање CIM инстанци из базе података и снимање у датотеку 61
График 3: Читање инстанце CIM модела из XML-а и из базе података 88
График 4: Снимање инстанце CIM модела у XML и у базу података 89
График 5: Измена инстанце CIM модела са праћењем и без праћења историје 90
График 6: Читање инстанце CIM модела из базе података и његова рестаурација 91

Списак прилога

Прилог 1: Исечак CIM UML дијаграма класа EQ профила 104
Прилог 2: Исечак CIM шеме базе података активног модела 105
Прилог 3: Исечак CIM шеме базе података историјског модела 106
Прилог 4: Исечак пресликавања класе на шему базе података, активни модел 107
Прилог 5: Исечак пресликавања класе на шему базе података, историјски модел 108
Прилог 6: Исечак алгоритма за поништавање појединачне измене 109

Списак уведених појмова

А

активна база 64
активна инстанца 64
активни модел 64
апстрактна табела 65

Г

група разлика
Difference Group 69

И

историјска база 66
историјска класа 80
историјски елемент 68
историјски модел 66

К

Комуникациони слој
API Communication Layer 81
конкретна табела 65

Н

наредне разлике
After Group of Changes 71
низ историјских елемената 68

П

пресек разлика
Slice 73
претходне разлике
Before Group of Changes 71

Списак скраћеница

ACID	Atomicity, Consistency, Isolation and Durability
ANN	Artificial Neural Network
AOP	Aspect Oriented Programming
API	Application Programmatic Interface
BASE	Basically Available, Soft Sate and Eventual Consistency
CA	Contingency Analysis
CASE	Computer-aided software engineering
CGMES	Common Grid Model Exchange Standard
CIM	Common Information Model
CIMODB	CIM Oriented Database
DACF	Day Ahead Congestion Forecast
DG	Difference Group
Diff	Difference file
DL	Diagram Layout
DN	Domain Name
DSL	Domain Specific Language
DW	Data Warehouse
DY	Dynamics
ENTSO-E	European Network of Transmission System Operators for Electricity
EQ	Equipment
ER	Entity Relationship
FC	Fault Calculation
FD	Forecasting Data
GE	Geographical Location
IEC	International Electrotechnical Commission
LF	Load Flow
LOC	Lines Of Code
MDA	Model Driven Architecture

MDD	Model-Driven Development
MDE	Model Driven Engineering
MDI	Model Driven Interoperability
MDSD	Model-Driven Software Development
MIC	Model-Integrated Computing
MOF	Meta Object Facility
MVC	Model-Viewer-Controller
NoSQL	Not Only SQL
NTC	Net Transfer Capacity
OLTP	On-line Transaction Processing
OMA	Object Management Architecture
OMG	Object Management Group
OOP	Object Oriented Programming
ORM	Object-Relational Mapping
PC	Personal Computer
PIM	Platform Independent Model
PL/SQL	Procedural Language/Structured Query Language
PSM	Platform Secific Model
RDBMS	Relational Database Management System
RDF	Resource Description Framework
SP	Stored Procedure
SQL	Structured Query Language
SSH	Steady State Hypothesis
SV	State Variables
SVM	Support Vector Machine
TC57	Technical Committee 57
TP	Topology
TPA	Transmission Planning and Analyzing software
TSO	Transmission System Operator
UCTE	Union of the Co-operation of Transmission of Electricity
UI	User Interface
UML	Unified Modeling Language
WG13	Work Group 13
XML	Extensible Markup Language

XSD	XML Schema Definition

1. Увод

Од средине двадесетог века електроенергетски системи налазе се у широкој употреби. У почетку, електроенергетски системи били су једноставни, изоловани системи. Како би надгледали стање електроенергетске мреже, њене улазне и излазне токове, оператери преносних система морали су да користе модел података којим би управљали потребним подацима. Прва европска електроенергетска асоцијација – Унија кооперације преноса електричне енергије (енгл. Union of the Co-operation of Transmission of Electricity – UCTE), установљена је 1951. године [32]. Од појављивања UCTE-а, производња електричне енергије, пренос, дистрибуција, аукцијско трговање и продаја на велико и мало полако је прелазило у надлежност одвојених компанија.

Након поделе производње, преноса, дистрибуције и продаје у електроенергетском сектору, могућност лаке размене оперативних података постала је изузетно важна, и то због све већег обима размене података која настаје заједно с разменом на тржишту електричне енергије. Према томе, до електроенергетске размене долази како између различитих компанија, тако и између одвојених држава. Значајна количина електричне енергије преноси се из једне области у другу. Чест је случај и да оператери преносних система купују електричну енергију за одређени сат током дана, а продају за неки други сат [14]. Да би се трговање електричном енергијом несметано одвијало и да би оператери преносних система могли на ефикасан и сигуран начин да управљају мрежом за коју су одговорни, потребна је поуздана и прецизна размена података између електроенергетских система.

Од појаве UCTE асоцијације, њен модел за опис електроенергетских мрежа, UCTE DEF, био је широко коришћен. UCTE DEF је модел првенствено орјентисан ка опису токова снага. Он носи податке само о основним, неопходним информацијама о чворовима, водовима и граничним чворовима електроенергетске мреже. Из тих разлога, тај модел може се окарактерисати као релативно једноставан. UCTE DEF је модел дизајниран тако да испуни захтеве европских компанија које су се доминантно појављивале средином двадесетог века и дуго је као такав задовољавао њихове потребе. Међутим, последњих деценија, његова једноставност показала се и као главни узрок његових ограничења: трансформаторске станице моделоване су са само једном сабирницом, док их у стварности може бити више; концепти потрошача, синхроне и асинхроне машине, нису дефинисани; ограничена је топологија мреже, на пример, уколико је потребно да се искључи једна компонента одређеног чвора, због структуре модела, оператер је приморан да искључи цео чвор.

У циљу превазилажења ограничења UCTE DEF модела, развој новог модела за опис електроенергетских мрежа покренут је 1999. године. У питању је Општи информациони модел (енгл. Common Information Model – CIM). CIM је модел који је још увек у развоју, а за његово одржавање задужена је Интернационална електротехничка комисија (енгл. International Electrotechnical Commission – IEC) [19]. У склопу сопственог

развоја, кооперације и координације, Европска мрежа оператера преносних система за електричну енергију (енгл. European Network of Transmission System Operators for Electricity – ENTSO-E) [32], наследник UCTE асоцијације, прихватила је CIM модел као препоручени за опис електроенергетских мрежа.

CIM је мрежно орјентисан модел, који на стандардизован начин пружа потребне информације за управљачке системе, мреже, програме и сервисе, а постоји и могућност његовог проширивања [19]. Недостаци који су уочени у UCTE DEF моделу, отклоњени су у CIM моделу. У односу на UCTE DEF модел, значајно је проширен скуп обухваћених концепата. CIM обухвата како физичке елементе електроенергетског система, као што су: електроенергане, трансформаторске станице, водове, чворове, прекидаче, тако и апстрактне елементе као што су дефиниције јединица мере: напона, снаге, отпора, и слично. CIM обухвата и концепте за опис дефиниција јединица мере (енгл. Domain – DN), опреме (енгл. Equipment – EQ), чворова (енгл. *Topology* – TP), географског положаја (енгл. Geographical Location – GE), приказа електроенергетских шема (енгл. Diagram Layout – DL), тренутних стања мреже (енгл. State Variables – SV), улазних података за основни прорачун токова снага (енгл. Steady State Hypothesis – SSH), као и информације о реаговању опреме на промене у мрежи (енгл. Dynamics – DY).

Развој CIM модела покривен је низом стандарда. Део CIM-а за преносне електроенергетске системе покривен је стандардом IEC 61970, издатим од стране IEC Техничког комитета број 57, Радна група 13 (енгл. Technical Committee 57, Work Group 13 – TC57, WG13). У тренутку писања овог рада, последња верзија стандарда била је 2.4.15, а верзија модела 16.25. Профил који ће бити коришћен у овом истраживању је Општи мрежни модел за размену (енгл. Common Grid Model Exchange Standard – CGMES) који се користи у преносним мрежама. Када се референцира појам CIM-а, у овом раду мисли се на стандард IEC 61970, профил CGMES, верзија стандарда 2.4.15, верзија модела 16.25. Сам CIM модел је специфициран путем Свеобухватног језика за моделовање (енгл. Unified Modeling Language – UML) [51].

CIM, на који се овај рад позива, заснован је на стандардима:

- IEC 61970-552: CIM XML Model Exchange Format,
- IEC 61970-301: Common Information Model (CIM) Base,
- IEC 61970-302: Common Information Model (CIM) for Dynamics Specification,
- IEC 61970-452: CIM Static Transmission Network Model Profiles,
- IEC 61970-453: Diagram Layout Profile,
- IEC 61970-456: Solved Power System State Profiles,
- IEC 61970-457: Common Information Model (CIM) for Dynamics Profile и
- IEC 61968-4: Application integration at electric utilities – System interfaces for distribution management – Part 4: Interfaces for records and asset management.

Низ правила и захтева дефинисан је у CIM стандарду како би била постигнута интероперабилност између различитих системских оператера (енгл. Transmission System Operator - TSO) унутар ENTSO-E асоцијације [51].

Према Институту за стандардизацију Србије и стандардима дефинисаним кроз *SRPS EN 61968-11:2014* [61] и *SRPS EN 61970-301:2014* [62] за дистрибутивне и преносне мреже, респективно, CIM се преводи као Заједнички информациони модел. У датом случају

енглеска реч *common* је преведена као „заједнички“, јер наведени стандарди покривају и дистрибутивне и преносне електроенергетске мреже. Међутим, за потребе овог рада, користи се термин Општи информациони модел, јер CIM модел у суштини представља генерички модел којег можемо прилагодити и проширити разним системима и разним наменама.

CIM дефинише метамодел за електроенергетске мреже, а инстанце CIM модела су модели конкретних мрежа. У овом раду CIM ће представљати почетни метамодел за приступ који ће бити предложен, а који се заснива на парадигми развоја заснованог на моделима (енгл. Model-Driven Development – MDD) [11], [30], [31], или, прецизније, развој софтвера заснованог на моделима (енгл. Model-Driven Software Development – MDSD) [8].

1.1. Предмет и проблем истраживања

Област истраживања предложеног рада односи се на развој и коришћење формализама и метамодела у области моделовања електроенергетских мрежа. У истраживању, од значаја ће бити формализми и начини организације података у области моделовања електроенергетских мрежа, док се у детаље семантике коју ти подаци носе неће улазити. CIM је један од заступљенијих метамодела за опис електроенергетских мрежа и представљаће основу за реализацију овде предложених истраживања.

Иако је у сталном развоју, CIM је постао широко прихваћен стандард у пракси. Постоји потреба да се CIM модел користи у електроенергетским функцијама, а самим тим и потреба да се чувају његове инстанце. Одатле следи главни предмет истраживања – проналажење методолошког приступа за смештање инстанци CIM модела у базу података, омогућавање праћења промена над инстанцама модела, као и рестаурација инстанци у неко од претходних стања.

Откривање могућих начина смештања CIM инстанци у базу података представља истраживачки проблем. Показује се да директно коришћење готових алата, који просто пресликавају CIM објекте у шему релационе базе података коришћењем тзв. објектно-релационог пресликавања (енгл. Object-Relational Mapping – ORM), не доводи до задовољавајућег решења. Ово посебно долази до изражаја када је потребно преузети инстанцу модела из базе података, како и када је потребно памтити промене над тим инстанцама и касније их рестаурирати у жељено стање.

С друге стране, коришћење парадигме релационог модела података и релационих система за управљање базама података намеће се као битан захтев који је у предложеном истраживању важно испунити [22], [46]. Он произилази из чињенице да инстанце CIM модела морају бити строго структуриране и морају омогућити да на што бољи начин пресликају уграђену семантику система у саму структуру шеме базе података. CIM модел састоји се од око 600 класа, укључујући и апстрактне и конкретне класе. Ако би свака класа била пресликана директно у једну дефиницију табеле релационе шеме базе података, један упит за читање једног скупа података морао би да референцира и све табеле које су добијене пресликавањем из апстрактних класа.

Обезбеђење праћења промена над инстанцама CIM модела представља обавезан захтев који овим истраживањем мора бити испуњен, јер у електроенергетским системима постоји стална потреба за ефикасним приступом историјским верзијама модела таквих система. При томе, наведени захтев не може бити реализован тиме што ће бити

обезбеђено меморисање комплетних инстанци модела у датим тренуцима времена, већ мора бити подржано меморисање искључиво измена које су настале, а у циљу обезбеђења ефикасног приступа историјским верзијама модела. Како се у пракси измене над CIM инстанцама спроводе путем целих група операција ради ефикасне размене измена над моделом, такав принцип може се искористити и за меморисање историјских верзија, које ће бити засновано на бележењу спроведених измена. Потребно је истражити могућност да се измене, извршене над елементима инстанци модела, аутоматски меморишу како за поједине елементе, тако и за групе елемената којима ти елементи припадају.

Обезбеђење могућности ефикасне рестаурације инстанци CIM модела такође је важна функција коју ово истраживање мора подржати, јер у пракси постоји потреба за враћањем инстанци модела у претходно стање. Да би захтевана рестаурација инстанце модела могла бити обављена на ефикасан начин, потребно је пронаћи у бази података минимални скуп елемената који је неопходно вратити у жељено претходно стање.

1.2. Циљ истраживања и очекивани резултати

Циљ истраживања је проналажење методолошког приступа и софтверског окружења који би омогућили олакшану реализацију рачунарски засноване програмске апликације за ефикасно складиштење CIM инстанци у релациону базу података, праћење измена над инстанцама модела и рестаурацију инстанци у жељено историјско стање.

Очекивани резултати истраживања су следећи:

- теоријски допринос: долазак до методолошког приступа који ће олакшати увођење CIM модела у софтверске алате за електроенергетске прорачуне,
- апликативни допринос: долазак до софтверског окружења с генераторима програмског кода које ће олакшати увођење CIM модела у софтверске алате за електроенергетске прорачуне и
- практични допринос: долазак до софтверске апликације за подршку складиштења преносних мрежа и управљање преносним мрежама, применом предложеног методолошког приступа и развијеног софтверског окружења, као и анализа примењивости предложеног приступа и софтверског окружења у практичним условима једног изабраног софтверског система.

Очекивани теоријски допринос обухвата методолошки приступ, који би се у овом случају односио на развој апликација за управљање инстанцама CIM модела. Овај методолошки приступ требало би да буде опште примењив и на друге моделе из истог проблемског домена. Очекује се да апликације које раде са инстанцама CIM модела подрже рад са релационим базама података, праћење историје измена инстанци модела и рестаурирање у жељена историјска стања.

Очекивани апликативни допринос обухвата креирање окружења са генераторима програмског кода. Развој генератора кода биће заснован на принципима метапрограмирања и MDD, ослањајући се на CIM модел. Производ рада генератора кода биће софтверска апликација, која ради са инстанцама CIM модела. Коришћењем спецификације модела, производи се програмски код који ради са инстанцама тог модела. Генератор кода треба да произведе софтверско решење које ће обухватити што

више случајева коришћења, који се очекују у оперативном раду. Производ добијен таквим генератором кода треба да буде исправан, чак и када дође до измена у улазном моделу [38]. Овим приступом очекује се постизање бржег развоја потребног софтвера, мањи број грешака и његово лакше одржавање [12], [57].

Очекивани практични допринос обухвата долазак до софтверске апликације за подршку складиштења преносних мрежа и управљање преносним мрежама на којима ће бити анализирана и проверена примењивост предложеног методолошког приступа, развијеног софтверског окружења и којима ће бити указано на искуства њихове практичне примене. При томе, није суштински важно за ово истраживање да ли ће бити у питању засебна софтверска апликација или проширење постојеће, изабране софтверске апликације. Сврха доласка до такве апликације је стварање услова за њену активну употребу у пракси.

Питање избора приступа за саму имплементацију програмског решења за смештање, праћење и рестаурирање CIM инстанци није од суштинске важности за само истраживање. Међутим, питање избора између имплементације засебне апликације или проширења постојећег софтверског решења мора бити решено у раним фазама овог истраживања. Предложена софтверска апликација представљаће компромис између наведена два приступа. Биће развијена засебна конзолна апликација, али која ће се у потпуности ослањати на библиотеке, у којима ће бити смештена сва програмска подршка потребна за рад са базом података, као и праћење и рестаурирање CIM инстанци. Практично, сва функционалност коју треба да подржи, била би смештена у одвојене програмске библиотеке. На тај начин, укључење овде предложеног решења у неко постојеће решење било би знатно олакшано.

Међу програмским апликацијама за рад са CIM инстанцама модела, планерски алати за планирање и анализу преносних електроенергетских мрежа (енгл. Transmission Planning and Analyzing software – TPA) уочавају се као могуће место примене овог истраживања. Неколико истакнутих компанија у домену развоја алата за управљање, планирање и анализу електроенергетских мрежа развијају своје TPA алате. Они се у већој или мањој мери ослањају на CIM модел за чување и размену података, као и за приказ вредности инстанци модела. Већина TPA алата користи CIM модел и као структуру података за електроенергетске прорачуне. Мана већине TPA алата је што се инстанце CIM модела чувају у основном XML облику, као датотеке на диску. Тиме је угрожена безбедност података, сигурност рада апликације, атомичност операција, конзистентност података. Могућност рада више апликација над истим подацима је упитна, не прати се историја промена и не постоји могућност рестаурирања инстанци модела у претходно стање.

Да би овде предложена решења неких од TPA алата била укључена у рад, први корак био би додавање развијених програмских библиотека које пружају подршку за рад са CIM орјентисаном базом података. Следећи корак био би прелазак са чувања CIM инстанци у XML формату у систему датотека на смештање инстанци у бази података. Последњи корак било би проширење датог TPA алата, тако да омогући приступ додатним функционалностима библиотека, као што су увид у историју измена и рестаурирање инстанци модела.

1.3. Методе истраживања

Опис методе истраживања представљен је по кључним фазама: развој методолошког приступа, моделовање шеме базе података за инстанце модела и за праћење стања инстанци, развој поступка за рестаурацију стања, развој генератора кода и примена у развоју софтверске апликације и анализа са оценом развијеног решења.

Предложени методолошки приступ мора бити систематичан, с јасно дефинисаним корацима које треба применити. Потребно је да буде теоријски заснован, односно изведен из општих, савремених метода развоја софтвера које се данас користе у софтверском инжењерству.

Како је CIM спецификован путем UML модела класа, та спецификација се може искористити и за креирање циљне шеме базе података. Како је CIM комплексан модел са много хијерархијских нивоа, потенцијални проблем проистиче из постојања великог броја апстрактних класа. Уколико би једноставно свака класа била пресликана у једну дефиницију табеле, упити над таквом базом података били би комплексни, јер би захтевали велики број спајања различитих табела и, потенцијално, перформансе извршења таквих упита биле би слабије. У оквиру новог методолошког приступа, потребно је обезбедити такве принципе пројектовања шеме базе података који ће омогућити једноставнији приступ потребним подацима.

Потребно је обезбедити одговарајућу логичку структуру шеме базе података која ће омогућити меморисање свих потребних података активних инстанци CIM модела. Такође, потребно је обезбедити меморисање историје измена, тако да велика историја измена не сме утицати на перформансе у оперативном раду, односно на операције које се спроводе над тренутним, активним стањем модела. Поред тога, потребно је пронаћи начин да се прате само неопходни подаци, а да се измене меморишу тако да обухвате само онолико елемената колико им је потребно.

За рестаурацију инстанце модела потребно је обезбедити могућност груписања измена, направљених над инстанцама модела. Уколико се у методолошком приступу предвиди могућност управљања групама измена, може се очекивати да би тражење потребних измена било лакше, него у случају када се врши претраживање свих измена, по свим елементима инстанце модела.

1.4. Задаци истраживања

Из перспективе постављеног циља истраживања, поред проучавања одговарајућих теоријских основа, идентификовани су следећи задаци:

- З1 – дефинисање методолошког приступа за развој програмске апликације која би омогућила смештање инстанци CIM модела у релациону базу података, праћење измена и рестаурирање у жељено претходно стање.
- З2 – дефинисање шеме базе података за смештање инстанци CIM модела.
- З3 – дефинисање шеме базе података за праћење стања инстанци CIM модела.
- З4 – дефинисање поступка за рестаурацију инстанци CIM модела.
- З5 – имплементација генератора кода који би олакшао имплементацију потребне апликације.

- З6 – имплементацију потребне апликације.
- З7 – анализа и оцена добијеног решења.

Начин реализације Задатка З4 зависиће од исхода реализације Задатака З2 и З3, јер он зависи од структуре података која ће бити дефинисана кроз Задатке З2 и З3. Задатак З6 зависи од задатка З5, јер се већи део његове имплементације добија кроз задатак З5.

1.5. Хипотезе истраживања

Из претходних истраживања која су описана у [22], [53] и [56], следи да за развој базе података као репозиторијума за инстанце комплексних модела као што је CIM не постоји једноставно и јединствено решење. Полазећи од Тукијеве максиме да је „проналажење питања често важније од проналажења одговора” [34], у овом случају полази се од питања: колико је оправдано да се CIM модел имплементира кроз релациону шему базе података и да CIM инстанце буду смештане у релационој бази података, узимајући у обзир све податке које CIM инстанце носе.

Одавде следи прва хипотеза истраживања:

Х1: Постоји могућност да се инстанце CIM модела сместе у релациону базу података, тако да не дође до губитка података.

Циљ истраживања је да се формулише методолошки приступ за генерисање релационе шеме базе података намењене за обезбеђење ефикасног смештања CIM инстанци, праћење промена над њима и, по потреби, враћање у неко од претходних стања. Као подршка овом истраживању, предлаже се и развој програма који би представљао примену описаног методолошког приступа.

Посебан захтев у функцији постављеног циља је да се предложи модел структуре података, такав да може омогућити чување како тренутних, тако и историјских стања CIM инстанци унутар једног релационог система за управљање базама података (енгл. Relational Database Management System – RDBMS). У циљу испуњења наведеног захтева, у предложеном раду уводе се појмови: активни модел и историјски модел. Активни модел описује тренутна стања CIM инстанци, док историјски модел описује претходна стања CIM инстанци. Развој активног и историјског модела треба да буде организован у две посебне фазе предложеног методолошког приступа.

Одавде следи друга хипотеза истраживања:

Х2: Могуће је обезбедити праћење промена над инстанцама CIM модела, смештених у релациону базу података.

Полазна мотивација за развој методолошког приступа за смештање CIM инстанци у релациону базу података је почетак примене новог модела за опис електроенергетских мрежа. У овом истраживању биће разматрано и питање подршке за прелаз са UCTE DEF модела на CIM модел за преносне мреже.

Да би у потпуности био омогућен рад са CIM инстанцама, потребно је развити и механизам који би омогућио праћење измена на одређеној инстанци CIM модела, као и рестаурацију инстанци у жељено или претходно стање. Потребан је методолошки

приступ који би описао поступак за моделовање базе података, такве да обезбеђује смештање како тренутних, тако и претходних стања CIM инстанци, праћење промена и рестаурирање инстанце у претходно стање.

Одавде следи трећа хипотеза истраживања:

Х3: Могуће је обезбедити рестаурацију инстанци CIM модела из релационе базе података у неко од изабраних стања.

Како је ENTSO-E прихватио CIM модел као препоручени модел, CIM је постепено постао широко прихваћен. Постоји доста радова који се баве конкретно CIM моделом, у којима је анализирана важност преласка са UCTE DEF модела на CIM модел [14]. Можемо очекивати да ће коришћење CIM модела значајно унапредити тачност, квалитет и поузданост размене између оператера преносних система [32]. Такође, уочено је да добити при увођењу CIM модела постају све бројније [33]. Развијени су електроенергетски алгоритми засновани на CIM моделу [25]. Поређени су типови описа преносних електроенергетских мрежа заснованих на CIM моделу [60]. Као последица тога код произвођача програма за електроенергетске системе јавила се потреба да се CIM укључи како у постојећа, тако и у нова програмска решења. CIM представља знатно обимнији и комплекснији модел у односу на UCTE DEF модел, што је један од главних разлога зашто његова примена у електроенергетским прорачунима, размени и складиштењу модела представља релативно сложен проблем који захтева детаљну анализу и развој посебног приступа.

Да би CIM могао боље и уз мање напора да се искористи у пракси, у оквиру апликативног дела истраживања треба развити софтверску апликацију за рад са инстанцама CIM модела, и омогућити методе за њихово смештање у део базе података који се односи на активни, а затим и у онај који се односи на историјски модел. Како је CIM модел у сталном развоју, важан захтев је обезбеђивање једноставног одржавања развијене апликације. Из тог разлога, у раду ће бити предложен развој генератора програмског кода који би омогућио аутоматску имплементацију апликативног програмског интерфејса (енгл. Application Programmatic Interface – API) за комуникацију са базом података. На тај начин, биће олакшано спровођење измена апликације у случају да дође до промена над CIM моделом. Овим би сложеност и количина посла потребна за развој и одржавање саме апликације била значајно смањена.

Поредећи UCTE DEF и CIM модел, уочено је да први има свега 4 типа података, док други има око 600 типова података и знатно комплекснију структуру. Из тог разлога, очекује се да би имплементација софтверске апликације, која би омогућила управљање CIM инстанцама, захтевала много више напора него имплементација апликације за рад са UCTE DEF инстанцама. Уколико би потребан програмски код за процедуре базе података и комуникациони програмски слој био развијан на класичан начин у којем тим програмера пише сваку појединачну линију кода, вероватноћа да дође до логичке грешке повећава се с порастом модела. Такође, с порастом модела, повећава се и потребно време за развој комуникационог програмског слоја, као и саме апликације. Одавде следи да је аутоматско програмирање, као парадигма за развој софтверских апликација, како је описано у [5] и [38], далеко ефикасније од класичног развоја. Ово је главни разлог за развој генератора кода, који би био коришћен за продукцију потребног програмског кода. Очекивано је да ће развијени генератор кода омогућити

ефикаснији развој шеме базе података, процедура унутар базе података, као и програмски слој за комуникацију с базом података.

Одавде следи четврта хипотеза истраживања:

X4: Развој генератора кода може да олакша развој CIM базе података и пратеће апликације.

Очекујемо да би развој методолошког приступа за развој CIM орјентисане базе података, која има подршку за праћење промена над базом и за њихову рестаурацију, дао одговор на питање: како остварити овде постављене хипотезе? Уједно, такав методолошки приступ допринео би и савладавању комплексности CIM модела. Последично, то би омогућило бржи развој апликација за рад са CIM моделом, као и релативно једноставан рад са историјским стањима инстанци.

Поред Увода, овај рад садржи преглед стања у области, анализу проблемског домена са освртом на могуће начине меморисања CIM инстанци, приступ развоју апликативног решења за управљање инстанцама CIM модела, опис развоја софтверске апликације за меморисање инстанци CIM модела и управљање инстанцама CIM модела, као и анализу и оцену развијеног приступа и софтверског решења.

2. Преглед стања у области

У овом поглављу дат је преглед радова који се ослањају на CIM, базе података и парадигму аутоматског програмирања, а који је у складу с предложеном темом истраживања планиране докторске дисертације.

2.1. Стандардизација CIM модела

Стандардизација CIM модела изузетно је важна, јер коришћењем стандардизованог модела података, различите програмске апликације, различитих компанија, могу међусобно да размењују податке без већих проблема. Као што се наводи у [10], *„Историја иза Општег информационог модела (CIM-а) је историја проналажења решења проблема који су одавно препознати од стране компаније које праве програмске алате за потребе оператера електроенергетских мрежа, као и од стране саме заједнице електроенергетских оператера.“* Било је потребно пронаћи јединствени модел за опис електроенергетских мрежа, такав да је независан од компаније која прави програмски алат за управљање његовим инстанцама. За опис семантике CIM-а, одабран је UML језик.

Координатор Радне групе 13, IEC технички комитет 57, за развој CIM стандарда, Чавадар Иванов у свом раду [14], говори о важности преласка са UCTE DEF модела на CIM модел. Иванов, у [14], наводи: *„тренутни UCTE DEF модел, и његов принцип размене података, показао се као недовољан за планерске алате, јер одређени подаци недостају и неки елементи мреже једноставно нису описани довољно детаљно.“* Оператери преносних система су одговорни за ефикасан и сигуран рад мреже којом управљају. Из тог разлога, њима су неопходни подаци о суседним мрежама, како би могли да реагују и на спољашње утицаје. Поред тога, дошло је до дерегулације електроенергетских система и повећања захтева тржишта. То је довело до повећаних варијација у производњи електричне енергије, како између држава, тако и између различитих области, у односу на два узастопна сата, па чак и на мањим интервалима. Све ово, довело је до још веће потребе за разменом података о мрежама. Како су могућности размене података, користећи UCTE DEF модел, ограничене, развој CIM модела постао је неопходност.

У свом раду [14], Иванов се залаже да:

- CIM формат треба да задовољи све захтеве електроенергетске заједнице и да мора следити IEC стандардизацију.
- Круцијални аспект имплементационе фазе модела је постепена миграција свих постојећих процеса размене података унутар UCTE асоциације. Очекује се да ће успешна имплементација и општа примена CIM стандарда, унутар UCTE заједнице, значајно унапредити тачност, квалитет и поузданост размене података међу различитим оператерима.

Такође, аутори Бритон и Де Вос, закључују да ће успешна имплементација и коришћење CIM модела значајно унапредити тачност, квалитет и поузданост размене између оператера преносних система [32]. Аутори уочавају и могуће ризике од додатних трошкова при промени модела, као и озбиљна ограничења да се сталним развојем модела скраћује животни век дате верзије CIM модела. Уколико се често мењају верзије модела, долазимо у ситуацију да се смањује корисност самог модела. Због тога, аутори предлажу измену улоге CIM модела, односно, заговарају идеју да се CIM не сматра више стандардом, него да му се да дизајнерска, генеричка, улога.

Аутори Бритон, Бровн, Моселеу и Бунда уочавају да однос предности и мана при увођењу CIM модела полако долази у равнотежу [33]. Поред тога, наглашавају да CIM може смањити трошкове, убрзати комплексне процесе, умањити потенцијалне грешке, као и ослободити инжењерско време од реализације активности управљања подацима. Аутори уочавају и да је стручно, мултилателарно прихватање CIM модела, као што је постигнуто у ENTSO-E асоцијацији, предуслов за коришћење предности CIM модела.

CIM је пројектован тако да омогућава рад са великом количином података и даје начине за релативно лако добављање података, њихово повезивање, записивање и размену између заинтересованих страна, како за тренутне, тако и за будуће потребе, при чему су многи случајеви коришћења још увек непознати. CIM пружа веома вредан модел за опис физичких карактеристика електроенергетских мрежа [13].

На основу ових радова, можемо закључити да је заједница корисника CIM модела значајно активна. Прихватање CIM модела од стране ENTSO-E асоцијације је одличан показатељ да CIM модел има своју будућност унутар електроенергетске заједнице. Одатле следи да развој софтвера за такав модел има своје оправдање и потенцијално тржиште.

2.2. Практична примена CIM модела

Како CIM модел улази у зрелу фазу развоја, тако се и више користи у пракси. У наставку текста биће презентована група референци која ће приказати могућа места примене методолошког приступа, предложеног овим радом.

У [25], аутори Лендак, Ердељан, и Поповић развијају алгоритам за проналажење каталога топологија у електроенергетском систему, описаног кроз CIM модел. У питању је истраживање у којем је потребно анализирати велике и комплексне електроенергетске мреже, оптимизовати дате мреже и користити их у електроенергетским прорачунима у реалном времену. У њиховом раду, CIM формат података може се у потпуности искористити за потребе алата који се користе у пракси.

Аутори Чапко, Ердељан, Вукмировић и Лендак у [16], решавали су проблем дељења дистрибутивне електроенергетске мреже. Почетна мрежа описана је путем CIM инстанци. Развијени алгоритам, назван Хибридни генетски алгоритам, може се извршавати у вишепроцесорском окружењу, и показује боље перформансе него обичан генетски алгоритам. Како је CIM изузетно детаљан модел, омогућен је рад са свим потребним деловима електроенергетске мреже.

Увођење паметних уређаја у електроенергетске мреже омогућено је рачунарским мрежама високих пропусних моћи и ниском ценом опреме. Услед релативне

приступачности потребне опреме, почеле су да се појављују паметне мреже (енгл. Smart Grid). Паметне мреже су довољно софистициране да могу обрадити велику количину радних процеса (енгл. workflow), са веома високим захтевима за ресурсима. Аутори Недић и Швенда у [47], представљају хибридни планерски (енгл. scheduling) алгоритам, који је резултат комбиновања централизованих и дистрибуираних планерских стратегија за алокацију радних процеса над дистрибутивним мрежама. Имплементација представљеног алгоритма такође је описана у раду. Статички подаци, над којима ради алгоритам, чувају се у CIM-компатибилном моделу података. Одавде се запажа да већ постоје разне варијације стандардног CIM модела, што говори о његовој заступљености у електроенергетским решењима.

Заступљеност CIM модела у пракси уочена је у раду [60]. Аутори Прадееп, Сешурају, Капарде и Јоши анализирају два основна типа описа електроенергетских мрежа: чвор-прекидач (енгл. node-breaker) и сабирница-грана (енгл. bus-branch). Оба типа могу се описати путем CIM модела, а основна разлика је у нивоу гранулације. У зависности од конкретне електроенергетске мреже, тип описа чвор-прекидач може садржати и неколико пута више података од типа описа сабирница-грана. Ово иде у прилог CIM моделу, који је пројектован да буде довољно робустан да може да подржи различите нивое грануларности [29].

У процесу увођења у употребу CIM модела за потребе описивања индијске електроенергетске мреже, аутори Равикумар, Капарде и Јоши сусрећу се са неколико проблема [21]. Значајније препреке представљају коришћење старих имена за одређене електроенергетске ресурсе, коришћење више апликација за управљање истим деловима мреже, као и постојање више начина за унос истих података због постојања предефинисаних идентификатора. Како би проблеми размене инстанци модела и именовања елемената били превазиђени, аутори предлажу проширивање модела информацијама које се користе у њиховим ранијим моделима. Аутори очекују да ће та проширења користити системским оператерима, компанијама које развијају програмске алате за управљање електроенергетском мрежом, електранама и регулаторима електроенергетских ситема.

Аутори Харгревес, Пантеа и Тејлор у свом раду [45] констатују да је за имплементацију паметних мрежа потребно повезивање уређаја од паметних мерача физичких вредности до апликација које управљају целим постројењима. У језгро свог решења за повезивање ових делова, аутори смештају управо CIM модел. Аутори уочавају да је интероперабилност вођена моделом (енгл. Model Driven Interoperability – MDI) на начин како би сви чиниоци електроенергетског система имали добит од детаљног и валидног описа мреже. На тај начин, лакше се уочава стање система, олакшава пословно планирање и подиже ниво ефикасности. Харгревес уочава сличну ситуацију на примеру Уједињеног Краљевства и потребе за повезивањем одвојених дистрибутера електричне енергије [44].

Овде представљени радови дају примере примене CIM модела у пракси. Ово су једни од типичних примера пројеката у којима би овде предложени приступ за смештање CIM инстанци могао наћи примену. Применом таквог приступа, ослањајући се на релационе базе података, можемо очекивати да је релативно лакше чувати конзистентност података користећи трансакционе измене. Различити корисници би могли једноставније да користе исте податке, већа је безбедност података јер постоји

могућност дистрибуирања базе података, а омогућено би било праћење измена које су извршене, као и враћање података на претходно стање.

2.3. Смештање CIM модела у базу података

Постоје различите стратегије складиштења CIM инстанци. Овде представљена истраживања могу значајно утицати на организацију података, као и на избор типа система базе података.

Један од два рада из ужег интересовања за потребе овог истраживања и из ове групе референци је [56]. Аутори Шулте, Бербнер, Стеинметз и Услар, у овом раду анализирају стратегије за смештање CIM инстанци. Аутори пореде релационе базе података и базе података организованих као Ресурсно дефинисано окружење (енгл. Resource Description Framework – RDF). CIM стандард, сам по себи, не нуди званични циљни модел базе података. Из тог разлога, тема рада била је да се пронађу предности и мане коришћења релативно нових RDF база података у односу на већ афирмисане и свеприсутне релационе базе података. Главни аргумент за коришћење RDF база података је да су веома дистрибуиране, не захтевају шему базе података и доста су скалабилне. Главни недостатак коришћења релационих база података, како је закључено у [56], јесте да се сви подаци морају унапред пресликати са CIM-XML формата на шему базе података. Међутим, како су релационе базе података много брже по питању перформанси, оне се ипак узимају као препоручљиве.

У другом раду [22] из ужег интересовања повезаног с овде предложеним истраживањем, аутори Равикумар, Капарде и Прадееп такође предлажу коришћење релационих база података, колоквијално SQL (енгл. Structured Query Language – SQL) база података. Уведен је термин CIM орјентисана база података (енгл. CIM Oriented Database – CIMODB), а сама шема базе креирана је коришћењем објектно-релационог пресликавања (енгл. Object Relational Mapping – ORM). У оба рада, [22] и [56], аутори сугеришу да се користе релационе базе података упркос растућој популарности графовски орјентисаних база података, односно *не-само-SQL* база података (енгл. Not Only SQL - NoSQL), те и RDF база података. Додатно, Равикумар и Капарде у [20] развијају посебну методологију како би повезали графовски орјентисану базу података и објектно орјентисану програмску апликацију, што захтева додатно истраживање и развој који се у овом раду надомешћују применом парадигме аутоматског програмирања.

Свеприсутно коришћење релационих DBMS, како Дарвен описује у [23], умањује потребу за коришћењем других технологија, као што су NoSQL базе података. Такав став заснива се на чињеници да су релационе базе података од свог појављивања, па све до данас, задржале изузетно високу поузданост, уз сасвим солидно поштовање устаљених стандарда. Под стандардима овде се мисли, пре свега, на подржаност SQL језика у готово свим релационим базама података.

Поједини аутори, Леавит у [46], као и Најак, Порија и Пооцари [6], доводе у питање оправданост коришћења NoSQL база података. Уочава се да NoSQL базе података почињу да заузимају приметан део тржишта база података. Аутори, међутим, скрећу пажњу и на потенцијално високу комплексност, упитну поузданост, конзистентност, односно стандардизацију, познавање конкретне технологије, као и ограничен број алата за рад са NoSQL базама података.

Аутори у [6] додатно наглашавају да NoSQL базе података не подржавају ACID захтеве:

- Атомичност (енгл. Atomicity) – серија операција која је таква да се или изврши у потпуности или се уопште не изврши;
- Конзистентност (енгл. Consistency) – све измене се врше само на дозвољени начин, уважавајући сва дефинисана ограничења;
- Изолација (енгл. Isolation) – означава видљивост трансакције другим корисницима система; и
- Трајност (енгл. Durability) – гаранција да ће резултати извршених трансакција остати очувани.

NoSQL подржавају BASE особине, односно:

- Углавном доступно (енгл. Basically Available) – систем гарантује одређени ниво доступности;
- Лабаво дефинисано (енгл. Soft State) – подаци не морају бити конзистентни и стриктно дефинисани у сваком тренутку;
- Крајња козистентност (енгл. Eventual Consistency) – подаци ће постати конзистентни у неком каснијем тренутку, уколико претходно не дође до нове промене података.

У раду [53], аутори Девић, Атлагић и Горечан, развили су базу података за смештање CIM модела, али без подршке за праћење промена над подацима и касније враћање на претходно стање. Недостатак прецизног методолошког приступа проблему и одсуство праћења историје захтева обимна преуређења и проширења датог рада, као и самог приступа. Свакако, предложена структура шеме базе може се искористити као једна од смерница за даљи развој.

Поред смештања инстанци CIM модела у релациону базу података, аутори Хе, Ванг, Жао и Ли у свом раду [59] анализирају и развој базе за потребе складиштења података (енгл. Data Warehouse – DW) ради каснијег формирања извештаја и анализе података. За те потребе, аутори су развили посебан метод који користе за смештање инстанци CIM модела у релациону базу података.

Примарно због резултата истраживања описаних у радовима [22] и [56], а уважавајући истраживања изнета у [23], [46], [53] и [6], у овом истраживању одлучено је да се користе релационе базе података. Главни разлози за одабир релационих база података јесу у томе што су оне афирмисане, свеприсутне, поуздане, стандардизоване и уважавају ACID захтеве.

2.4. Примена парадигме аутоматског програмирања

Када се пројектује програмска подршка за комплексне моделе, као што CIM модел са око 600 међусобно повезаних класа свакако јесте, онда се мора водити рачуна и о потребном времену развоја. У овом тексту ће бити дат преглед радова који користе парадигму аутоматског програмирања у циљу смањења времена потребног за развој и повећања поузданости добијеног кода.

Аутоматско програмирање, које је блиско парадигми метапрограмирања, увелико се користило у бројним пројектима од самог почетка развоја софтверског инжењерства [5], [38]. Како је Кос у [5] навела: *„... како бисмо омогућили да машина помогне при програмирању [...], покушали смо да апстрахујемо кораке, и укључимо их у више програмске језике. Ово је довело до појаве: интерпретера, асемблера, копајлера и генератора – програма писаних да би писали друге програме, односно, аутоматског програмирања.“* Леви у [38] проширује ову тезу: *„Метапрограмирање, дефинисано као писање програма који пишу потребне програме, представља основни метод за снижење цене развоја програма и повећање његовог квалитета.“*

Аутори Аткинсон и Кухне наводе у [11] да су истраживања још од појаве рачунарских наука увек тежила да повећају ниво апстракције на којем програмери пишу програме. Још од појаве првих компајлера, програмери су почели да пишу шта машина треба да ради, а не како да ради. Развој вођен моделом (енгл. Model Driven Development - MDD) природан је наставак тог тренда. Као основна мотивација наводи се:

- краткорочно унапређење продуктивности програмера повећањем способности реализације веће количине функционалности и
- дугорочно унапређење продуктивности програмера, тако што се смањује брзина којом релализоване функционалности постају застареле.

У [11] се дефинишу концепти које би једна MDD структура требало да подржи:

- Концепте доступне за стварање модела и правила које регулишу њихову употребу.
- Нотацију, која се користи за опис модела.
- Опис, како елементи модела презентују елементе из стварног света, укључујући програмске елементе.
- Концепте који олакшавају динамичко проширење концепта модела, нотацију модела, укључујући и изведене моделе.
- Концепте који олакшавају размену концепата и нотација модела, укључујући и изведене моделе.
- Концепте који олакшавају пресликавање модела на друге елементе, укључујући и програмски код онако како је дефинисано од стране корисника модела.

Традиционално, инфраструктура MDD сачињена је од четири нивоа апстракције. Сваки ниво, осим највишег, представља инстанцу вишег нивоа. Нивои су:

- M0 – представља конкретне податке корисника. Нпр. Милан, 5, 6 и слично.
- M1 – представља модел за податке у M0. Модели база података дефинишу се на овом нивоу. Нпр. Студенти и Оцене.
- M2 – је модел који се користи за дефинисање M1. Зато што је M2 модел за модел M1, он се назива метамодел. Нпр. UML [51]. Модели података дефинишу се на овом нивоу.
- M3 – садржи информације које се користе за дефинисање M2. Овај ниво зато представља метаметамодел, односно обухвата метаконцепте, путем којих се могу градити метамодели нивоа M2 (енгл. Meta Object Facility – MOF).

Ова вертикална, четворослојна инфраструктура обично се назива структура *лингвистичких* метамодела [11]. Веома је једноставна за дефинисање нових модела као инстанци М2 нивоа. Иако се показао као изузетно добра основа за прву генерацију MDD технологија, ова традиционална инфраструктура, како је уочено у [11], не даје одговоре на четири кључна проблема:

- Не постоји објашњење како су ентитети повезани са стварним светом. Да ли инфраструктура себи прилагођава стварни свет или он лежи ван ње?
- Инфраструктура подразумева да су све везе међу елементима истог типа. Да ли је ово валидан приступ или је потребно правити разлике?
- Не постоји одређен принцип за одређивање на ком нивоу треба да се нађе неки елемент. Да ли је довољно да прављењем инстанци дефинишемо границе између нивоа? Ако да, која су правила?
- Постоји склоност да се користи метамодел за дефинисање предефинисаних концепата. Да ли је могуће дефинисати их на неки други начин и користити унутар одређене архитектуре?

Како би наведени проблеми били превазиђени, предлаже се прелазак на *онтолошко* метамоделовање. Онтолошки метатипови јасно раздвајају нивое апстракције и користе се за њихово дефинисање. Сваки онтолошки ниво представља инстанцу вишег онтолошког нивоа. Сваки онтолошки ниво има два лингвистичка нивоа, Л0 и Л1. При томе, Л0 и Л1, представљају М0 и М1 из лингвистичког метамодела, респективно [11]. Могу постојати следећи онтолошки нивои:

- О0 – представља конкретне податке. Нпр. Милан. „Милан“ је лингвистички ниво Л0, који представља инстанцу лингвистичког нивоа Л1 – објекат.
- О1 – представља тип података. Нпр. студент основних студија. „Милан“ је онтолошка инстанца студента основних студија. А студент (Л0) представља инстанцу лингвистичке класе (Л1).
- О2 – може дефинисати даље апстраховање. Нпр. студент факултета. Студент основних студија, „Милан“, студент је на факултету. А студент факултета (Л0) представља инстанцу лингвистичке класе (Л1).
- О3 – може дефинисати даље апстраховање. Нпр. студент универзитета.
- О4 – може наставити уопштавање. Нпр. академски грађанин. Студенти су онтолошка инстанца академских грађанина. А академски грађани (Л0) представљају инстанцу лингвистичке метакласе (Л1).
- О5 – последњи ниво апстракције, представља класификацију која се описује. Нпр. класификујемо академски систем.

Онтолошки метамодел може имати произвољан број нивоа. Одатле следи да се могу уводити додатни нивои класификације. На овај начин аутори Аткинсон и Кухне [11] предлажу скуп концепата које би MDD требало да подржава.

Аутори Безивин и Гербе у [30] указују на еволуцију у примени методологија. Објектно орјентисана архитектура (енгл. Object Management Architecture – OMA), дефинисана од стране Објектне управљачке групе (енгл. Object Management Group – OMG), омогућила је да се замени претходна процедурално орјентисана парадигма новом, објектно орјентисаном парадигмом (енгл. Object Oriented Programming – OOP). Наставак тог

тренда је, како аутори кажу, још радикалнија промена – прелазак на трансформације модела. Конкретан пример је методологија архитектуре вођене моделом (енгл. Model-driven Architecture – MDA), предложена од стране OMG. Како је OMG група задужена за одржавање UML језика, ово се може сматрати предлогом који може имати значајног утицаја у информационим технологијама. Главни закључак је уочавање фундаменталне промене, где се модели више не користе само за документовање програма, него и директно – за развој потребног алата.

MDA, MDD, као и инжењерство вођено моделом (енгл. Model Driven Engineering – MDE) међусобно су блиски приступи. MDD је усмерен ка коришћењу модела за развој програма, док MDA даје смернице за дефинисање система путем платформски независних модела (енгл. Platform Independent Model – PIM). MDA је специфициран од стране OMG консорцијума и примарно покрива дефинисање модела и њихове трансформације [8]. Нешто шири опсег од MDA има MDE, где се комбинују процеси и анализе са архитектуром, а типичан резултат је сам модел, како наводи Кент у [55]. Платформски независан модел (PIM) користи се за добијање платформски зависног модела (енгл. Platform Secific Model – PSM) путем низа трансформација [15].

Како се наводи у [57], аутори Нгујен, Кафмола и Ричта наводе да генератор кода користи опис програма високе апстракције да би произвео програмски код ниже апстракције. Аутори су овај приступ искористили при стварању свог наменског језика, за реализацију жељених мрежних (веб) сервиса, следећи принципе развоја заснованог на моделима (MDD).

Аутори Хенторне и Тилевич у [12] уочавају да су комерцијални генератори кода постали саставни део савременог развоја програмских апликација. Користе се да би се олакшао развој бројних компоненти које су напорне за писање и осетљиве на грешке. Упркос томе, аутори наглашавају да нису увек генератори кода адекватни за одређене задатке. Као главне недостатке генератора кода наводе:

- Доследност конвенцији – поједини генератори кода на пружају довољно подесивих параметара да задовоље жељену конвенцију.
- Генерисани код не даје увек жељену функционалност.
- Генерисани код не може се увек извршавати у конкурентном окружењу.

Као решење за наведене недостатке, аутори у [12] предлажу коришћење анотација дефинисаних кроз аспектно орјентисано програмирање (енгл. Aspect Oriented Programming – AOP). На тај начин аутори постижу бољу функционалност генерисаног кода.

У [17], аутори Кунду, Саманта и Мал такође се ослањају на MDD за развој свог пројекта, користећи модел дефинисан у UML језику као полазну основу. Даље, уочавају да се генерисање кода на основу UML-а углавном ограничава на добијање структура, пресликаваних са UML дијаграма класа. Аутори предлажу да се оде корак даље и да се искористе дијаграми секвенци као помоћ при генерисању кода. Тиме се аутоматски добија имплементација позива функција, контрола извршења (алгоритам), као и интеракција међу компонентама. На тај начин аутори су успели да 48% линија програмског кода (енгл. Lines Of Code – LOC) добију користећи свој генератор кода. Као главне замерке оваквог приступа аутори наводе:

- код, генерисан на основу дијаграма секвенци, није обавезно комплетан, јер нису сви подаци који описују дато понашање описани кроз дијаграме секвенци;

- код није генерисан за све методе класа, него само за класе описане кроз дијаграме секвенци;
- програмер мора да реши потенцијалне проблеме, који се могу јавити у добијеном програмском коду.

У својој емпиријској студији, аутори закључују да је добијени генератор кода најефикаснији за добијање контролних класа у програмском обрасцу модел-приказ-контрола (енгл. Model-Viewer-Controller – MVC).

Аутори Аблонскис и Немураите у [37] развили су метод који детектује трансформације из модела у програмски код. Развијени метод може се искористити за: а) уочавање мањих имплементационих образаца коришћењем концептуалног модела (архитектура система); б) основне трансформације за поређење програмског кода и програмског модела; као и за в) поређење програмског модела и појединих делова кода који су ручно писани. Ова техника искоришћена је као део методологије за аутоматско креирање конфигурабилног генератора кода, који је намењен за генерисање програмског кода у случајевима када домен примене није у потпуности познат. Такође, може се искористити и када се платформа, на којој се извршава програм, мења толико брзо да је њен стабилни период краћи од времена потребног за класичан развој програма.

У [1], аутори Армонас и Немураите констатују да се у информационим наукама нове технологије појављују готово сваког дана, а релационе базе података актуелне су више од 30 година. Проблем који се уочава је чињеница да свега неколико комерцијално-пројектантских алата нуди потпуно генерисање релационе шеме базе података. Из тог разлога, аутори предлажу метод за генерисање комплетних модела релационе шеме базе података, а на основу концептуалног модела. У овом случају, користи се MDA, где се метамодел, дефинисан кроз UML, трансформише према задатим обрасцима да би се добио код потребан за циљану базу података.

Генератор кода, који су описали Фертаљ, Калпић и Морнар у [35], настао је као резултат покушаја да се унапреди развој интерактивне програмске апликације, која се ослањала на базу података. Развијен је посебан језик специјално за ту сврху. Коришћењем тог језика специфициране су процедуре које се користе за генерисање кода. Наравно, код се генерише на основу шеме базе податка. Основни кораци рада генератора су: узимање у обзир циљаног програмског језика и релациони DBMS; анализа шеме базе података; генерисање класа, релација и форме за приказ података и, на крају, генерисање саме програмске апликације.

Основни циљ, који су аутори Антовић, Влајић, Милић, Савић и Станојевић поставили у [24], био је да се уоче односи између: UML дијаграма интеракција, UML дијаграма класа и жељеног корисничког интерфејса (енгл. User Interface – UI). При дефинисању захтева, мора се узети у обзир следеће: како дијаграми интеракција описују интеракције између корисника и система; како се имплементира кориснички интерфејс; које функционалности програмер жели да омогући. У својој студији, аутори су развили модел који повезује поменуте целине. Аутори су развијени модел искористили у алату који трансформише инстанцу модела у извршни програмски код. У [24] наглашава се да развој програмских апликација представља серију интелектуалних и техничких активности изведених од стране високо образованих инжењера. Из тога постаје јасно да цена апликација увелико зависи од времена и напора потребног за њихов развој. Као предности генерисања кода наводи се:

- смањење логичких грешака,
- смањење времена развоја,
- смањење цене развоја,
- доследност добијеног кода,
- повећање квалитета,
- веће задовољство крајњег корисника,
- доследна примена програмских образаца,
- поновна искористивост кода и
- виши ниво продуктивности.

Као главне мане наводе се:

- генерисање кода није увек примењиво,
- генераторе кода је тешко направити,
- уочавање ситуација где би могло да се користи генерисање кода,
- утицај на развојни процес,
- висока почетна цена,
- дужа фаза пројектовања,
- управљање конфигурацијама при генерисању,
- подударност (компатибилност) са остатком програмског кода и
- утицај на коришћење других алата.

Аутори рада [24] наводе да су око 75% линија програмског кода (енгл. LOC) добили користећи свој генератор кода. Поред тога, цео пројекат је урађен за четири месеца, што је много мање времена од процењених дванаест месеци.

У [36], аутори Ма, Јанг и Абрахам користе MDE приступ да развију своју сервисну апликацију, а као додатак користе се обрасци да би се добио жељени програмски код. Позивајући се на MDE, аутори наглашавају да: више времена остаје за анализу пословних захтева; смањује се време потребно за писање програмског кода; и коначно, долази до повећања продуктивности јер се смањује време неопходно за писање програмског кода.

Аутоматско програмирање и подизање на више нивое апстракције у програмирању једна је од основних стратегија у развоју програмирања. При развоју софтвера, није циљ уложити што више напора у његов развој, него постићи жељени циљ са што мање напора, одржавајући висок квалитет добијеног решења. Људи су склони грешкама, што посебно долази до изражаја након вишечасовног рада на пројектима који захтевају висок ниво концентрације. Аутоматизација развојног процеса у софтверском инжењерству има за циљ растерећење индивидуе, повећање продуктивности, смањење грешака при раду и самим тим повећање квалитета коначног производа.

Како бисмо постигли наведене циљеве, при развоју овог методолошког приступа биће примењене неке од техника аутоматизације развојног процеса. Развој вођен моделом (MDD) спада у ту групу. Како је CIM модел дефинисан кроз UML модел класа, самим тим формиран је модел са јасно дефинисаном нотацијом која регулише његову употребу.

Унапред су дефинисани елементи CIM модела који представљају елементе из стварног света. Оно што је потребно самостално дефинисати је како даље пресликавати CIM модел у циљу доласка до жељеног програмског решења. За те потребе мора јасно бити дефинисан методолошки приступ који ће описати кораке који уважавају правила CIM модела, потребне структуре података и, на крају, поступак производње циљног софтверског решења.

Ослањање на релационе базе за смештање података, стратегија је која се примењује у софтверском инжењерству почевши од осамдесетих година XX века, а врло интензивно од деведесетих година, па до данас. Поузданост и сигурност релационих база података су особине које су се много пута показале као круцијалне при одабиру начина за смештање података, што илуструју и радови наведени у овом поглављу. Међутим, како се наводи у [1], не постоји много комерцијалних алата који нуде потпуно генерисање релационе шеме базе података. Ова чињеница је додатна смерница која оправдава потребу аутоматизације развојног процеса софтвера који је предмет овог рада. Очекује се да ће за добијање коначног производа коришћењем предложеног методолошког приступа бити потребан развој наменских алата за поизводњу потребног програмског кода.

2.5. Алати за аутоматизацију развоја програмских апликација

Домен програмирања који се бави развојем алата и метода за пројектовање и имплементацију програмских апликација (енгл. Computer-aided Software Engineering – CASE), блиско је повезан са парадигмом аутоматског програмирања. CASE алати служе за аутоматизацију развојног процеса, помоћу којих се израђује дизајн пројектног решења, који се потом користи при имплементацији. На тај начин, постиже се бољи квалитет и ефикасност при развоју програмског пројекта [2].

Према Кејсу [2], прави CASE алат требало би да поседује:

- графички приказ који пружа лак приступ свим деловима CASE алата,
- основни комплет развојних алата, као што су: графички и текстуални едитори, графичке представе елемената за моделовање, валидацију модела и могућност поновне искористивости креираних спецификација,
- основни комплет управљачких алата попут: документовања пројекта, процене потребног труда, распореда рада, прављења извештаја и оцене квалитета,
- систем помоћи који пружа приступ документацији CASE алата,
- подршку за снимање пројекта, укључујући праћење верзија пројекта и
- могућност повезивања са додатним алатима које би кориснику били од помоћи при развоју.

CASE алати су посебно корисни за организацију и контролу развоја софтвера, поготово за велике, комплексне системе, који укључују много различитих компоненти и људи који раде на датом пројекту. Коришћењем CASE алата омогућава се развојним инжењерима, дизајнерима, тест-инжењерима, пројектантима, пословођама и управницима да имају заједнички увид у развој пројекта и све његове фазе. На

сличном принципу заснива се и рачунарство интегрисано са моделима (енгл. Model-Integrated Computing – MIC), у којем се на основу модела праве програмска решења за одређени домен [4].

Према Фугети [3], CASE алати могу се поделити у следеће групе:

- едиторски: графички или текстуални;
- развојни: писање кода и дебаговање, укључују програмске библиотеке, компајлере, дебагере, интерпретере, претпроцесирање, генерисање кода, и слично;
- верификациони и валидациони: укључују алате за статичку и динамичку анализу кода, поређење кода, емулацију и симулацију, провера исправности кода, тестирање програма;
- конфигурациони: укључују алате за контролу конфигурације параметара и верзионисање, праћење коришћених библиотека;
- за управљање пројектом: укључују алате за процењивање трошкова, планирање, организовање састанака и слично;
- остало: одржавање веб страница и других презентација.

CASE алати покривају веома широк домен развоја софтвера, међутим, посебно је значајно осврнути се на генерисање кода. Потребно је генерисати код за веома специфичну намену, без потребе за већом конфигурацијом, због постојања једног модела који користимо. За разлику од тога, CASE алати, који имају подршку за генерисање кода, обично су намењени за ширу, генеричку примену. Модели са којима се ради најчешће су независни један од другог. Ипак, висока специфичност генерисаног кода, производ је развоја наменског генератора кода. С друге стране, коришћење општег CASE алата за добијање веома специфичног кода захтевало би веома високу конфигурабилност самог алата, која у постојећим CASE алатима углавном није омогућена.

Без обзира на уопштеност CASE алата, искуства при генерисању кода у тим алатима утичу на развој потребног генератора кода. У [27], аутори Луковић и остали развили су један алат вођен моделом (MDD) под називом IIS*Case, који покрива већину компоненти које један CASE алат, према Кајсу [2], треба да садржи. IIS*Case на основу PIM омогућава генерисање комплетног програма, почевши од графичког интерфејса, преко трансакционих структура за пренос података, до основних функционалности за рад са базом података. На тај начин, PIM се аутоматски трансформише у оперативни програм са основном пословном логиком. За потребе овог истраживања, потребно је да генератор кода обезбеди шему базе података организовану на начин који ће бити описан у Поглављу 4, и одговарајући код за приступ бази података.

У [28], аутори проширују IIS*Case увођењем наменског језика (енгл. Domain Specific Language – DSL), под именом IIS*CDesLang. Наменски језици (DSL) развијају се за потребе специфичног програмског домена [41]. У овом случају, провера формалне исправности креирања платформски независног модела (PIM) један је од главних разлога за увођење новог DSL језика. На тај начин, омогућава се писање текстуалног језика којим се моделује PIM за информационе системе. Обично се шеме база података описују дијаграмима са ентитетским релацијама (енгл. Entity Relationship – ER). Међутим, један од главних недостатака ER је мањак семантике [26]. Сходно мишљењу

аутора рада [28], у циљу превазилажења наведеног недостатка, предлаже се коришћење парадигми MDD и DSL. Основна идеја је да се омогуће потребни концепти за моделовање PIM-а на нивоу апликације. IIS*CDesLang специфициран је и путем графичког окружења за креирање атрибутивних граматика, под именом VisualLISA [40], [49].

Аутори рада [48] предлажу наставак развоја IIS*Case алата. У [48] даје се увид у имплементацију алгоритама за трансформацију ограничења, дефинисаних на PIM нивоу, у релациони модел података, а потом и у извршни SQL код. Подршка је остварена за неколико платформи, па је велика пажња посвећена трансформацији PIM у PSM.

У [39], аутори користе MOF спецификацију да би дефинисали концепте потребне за IIS*Case алат. Један од главних разлога за овакав приступ је добијање формалне софтверске документације. Такође, користећи PIM, омогућено је и генерисање компоненти за тестирање извршног кода.

Аутори у [7] настављају рад на IIS*Case алату, додајући могућност генерисања пословне логике, која представља функционалности специфичне за конкретну програмску апликацију. Како постоји мањак специјализованих алата за такву намену, предлаже се увођење језика специјализованог за домен примене (DSL) под именом IIS*CFuncLang. Сврха увођења овог језика је омогућавање дефинисања посебних, тзв. „нетипичних“ функционалности на PIM нивоу.

Са ограниченим ресурсима, ограниченим временом и бројем људи, да бисмо имплементирали програмско решење које се очекује као крајњи производ предложеног методолошког приступа, неопходно је аутоматизовати развојни процес. Како би аутоматизација била могућа, неопходно је развити алат који ће то омогућити. У овом случају, алат за аутоматизацију би представљао генератор програмског кода, чији производ ће бити програмски код који поштује задати CIM модел. Планирани генератор кода би представљао CASE алат са суженим скупом функционалности. Слично као у [27] и [28], генератор кода би се ослањао на MDD, где би CIM модел имао улогу PIM-а и њиме би се водио развој апликације. Као што је у [48] посвећена велика пажња трансформацији PIM у PSM, односно, трансформацији у релациони модел података и потом у извршни SQL код, слична функционалност очекује се и у овом раду. За разлику од радова описаних у [7], [28], [39], [40] и [49], где су аутори развијали и DSL, развој посебног DSL-а није планиран за потребе овог истраживања. Развој подршке за DSL био би неопходан уколико се појави потреба да генератор кода производи додатне компоненте програмске апликације са више „нетипичних“ функционалности.

Основна намена предложеног генератора кода била би производња програмског кода за потребе предложеног методолошког приступа. Без графичке или текстуалне представе, генератор кода би се у потпуности ослањао на већ дефинисани CIM модел. Потенцијалне конфигурације алата биле би сведене на подешавање избора које компоненте кода би биле подвргнуте генерисању, тј. производњи. По угледу на ауторе у [7], основни циљ предложеног алата биће производња програмског кода који задовољава потребе дефинисане кроз предложени методолошки приступ. Конкретно, планирани генератор кода ће као свој улаз имати CIM модел. Очекивани производ планираног генератора је програмски код који ће представљати крајњи производ предложеног методолошког приступа.

2.6. Закључак поглавља

У овом поглављу дат је преглед стања у области. Стандардизација CIM модела представљена је првенствено кроз рад Радне групе 13, IEC техничког комитета 57 за развој CIM стандарда. Такође, анализирани су и радови аутора који су утицали на правце развоја CIM модела.

Практична примена CIM модела представљена је кроз групу референци која је битна са становишта могућих места примене методолошког приступа, који ће у раду бити предложен.

Представљене су различите стратегије складиштења CIM инстанци. Наведена истраживања значајно ће утицати на организацију података, као и на избор типа система базе података.

Коначно, примена парадигме аутоматског програмирања, као и алати за аутоматизацију развоја програмских апликација, могу се искористити као смернице како да се са ограниченим ресурсима у људству и времену имплементира неко релативно комплексно решење.

3. Проблемски домен и анализа могућих начина меморисања инстанци CIM модела

CIM модел, због своје сложености, можемо уврстити у класу великих и комплексних модела. Из тог разлога, процена је да би било потребно много времена да се развије база података и програмски код за рад са базом података путем простог писања потребног кода. Због тога је развој генератора кода потребан у циљу обезбеђења подршке процесу имплементације CIM модела унутар RDBMS.

3.1. Основни појмови

Једна од основних намена CIM модела је размена података између чланова ENTSO-E асоцијације. Циљ је да оператери преносних система могу да размењују инстанце модела електроенергетских мрежа, користећи програмске апликације различитих компанија. Како то представља неопходну пословну активност, овде ће бити представљен део унутрашње структуре CIM модела, као и процес размене података који је дефинисан стандардом [19].

Основни формат чувања CIM модела је XML датотека (енгл. Extensible Markup Language – XML). XML је спецификован кроз шему XML-а (енгл. XML Schema Definition – XSD). Сами подаци подељени су у девет профила, који чине у пракси и девет XML датотека:

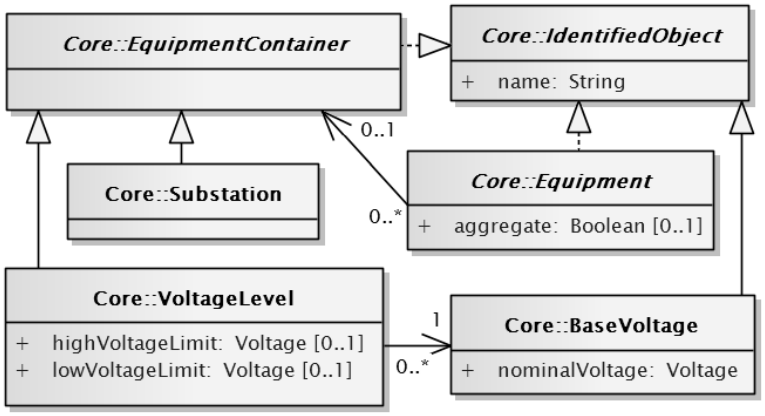
- Опрема (енгл. Equipment – EQ): описује статичке податке, електроенергетске уређаје који чине мрежу.
- Гранична опрема (енгл. Boundary EQ – BE): описује опрему на крајевима мреже, тј. опрему за повезивање две мреже.
- Топологија (енгл. Topology – TP): описује чворове мреже и садржи резултате тополошког процесирања.
- Гранична топологија (енгл. Boundary TP – BT): описује граничне чворове мреже, односно чворове који повезују две мреже.
- Географски положај (енгл. Geographical Location – GE): физичка локација поједине опреме.
- Приказ електроенергетских шема (енгл. Diagram Layout – DL): омогућује дефинисање шематског приказа мреже.
- Променљива стања (енгл. State Variables – SV): садржи резултате добијене на основу прорачуна токова снаге и обузвата тзв. излазне податке.

- Стабилна стања (енгл. Steady State Hypotesis – SSH): садржи динамичке податке опреме који служе као основа за порачун токова снага и обухвата тзв. улазне податке.
- Информације о реаговању опреме на промене у мрежи (енгл. Dynamics – DY).

Подаци из свих девет датотека представљају једну *комплетну* инстанцу CIM модела.

Један од важнијих разлога зашто је CIM модел издељен у девет профила је зато што учесталост измена на различитим деловима CIM модела није иста. Промена у EQ пакету настаје када долази до проширења мреже. Оваква врста промене настаје обично месечно или сезонски. SSH профил је замишљен као допуна EQ пакета. SSH профил се веома често мења, обично сваког сата, али може и чешће. Подаци који се бележе у SSH профилу издвојени су управо да би се EQ пакет могао изнова користити док не дође до промена у структури мреже. TP и SV профили представљају резултате прорачуна, тако да им је учесталост измена иста као и за SSH профил.

CIM је хијерархијски модел којег чине апстрактне и конкретне класе. Физички аспекти електроенергетског система, као и јединице мере које описују један такав систем, пресликавају се у опис CIM модела који је дефинисан путем UML модела класа. Стања електроенергетског система измерена у одређеном тренутку бележе се кроз инстанце CIM модела. Апстрактне класе користе се да упросте комплексност система. Оне дефинишу опште атрибуте и релације и раздвајају опште компоненте система. С друге стране, стварне (конкретне) компоненте система описане су конкретним класама које наслеђују већи део својих атрибута и асоцијација од апстрактних класа. Конкретне класе, последично, зависне су од апстрактних класа, што је приказано на Слици 1, која представља извод (део) CIM модела. Потребно је нагласити да појединачне инстанце модела садрже само конкретне класе. На Слици 1, Контејнер опреме (енгл. Equipment Container) представља апстрактну класу, док Трансформаторска станица (енгл. Substation), Напонски ниво (енгл. Voltage Level) и Основни напон (енгл. Base Voltage) представљају конкретне класе. Како је истраживачки интерес у овом раду првенствено усмерен на односе ових класа, њихово значење овде неће бити анализирано. Ради бољег увида у комплексност CIM модела, у Прилогу 1 је дат нешто шири исечак модела у односу на Слику 1.



Слика 1: CIM UML дијаграм класа, исечак из EQ профила

Сваки елемент, објекат класе, унутар једне инстанце CIM модела мора имати јединствени идентификатор. Овим се осигурава да се елементи једне инстанце модела могу разликовати. Међутим, треба обратити пажњу да не постоји једна класа која дефинише идентификатор и коју наслеђују све остале класе.

Измене, направљене над инстанцама модела, размењују се путем *датотека разлика* (енгл. Difference file – Diff) које садрже информације само о томе шта је измењено, односно шта је додато и шта је избрисано из инстанце модела. Када једна страна направи одређене измене над својом инстанцом модела, потребно је да креира и датотеку разлика. Датотека разлика користи се како би остале стране којима је дата мрежа од интереса, примениле исте измене над својим инстанцама модела дате мреже [19]. Ово је једна од главних предности CIM модела, јер за одржавање тренутних стања инстанци модела мреже није неопходно размењивати целе инстанце модела, него само промене до којих је дошло [14], [32]. Датотеке разлика такође су структуриране путем XML формата и попут инстанце модела садрже само конкретне класе.

У овом раду, скуп података (енгл. table, data set) представља колекцију објеката-података (енгл. data objects) који се негде назива и торком, појавом, записом, опсервацијом или случајем. Ради једноставности израза и због боље аналогије са базама података уместо појма објекат-податак, појмови скуп података или табела података биће коришћени у тексту овог елабората.

Обележје (атрибут, енгл. attribute, column) је својство или карактеристика објекта које може варирати од објекта до објекта и варирати у времену.

Скала мерења је правило (функција) које повезује нумеричку или симболичку вредност са обележјем неког објекта. Поступак мерења је примена скале мерења у циљу повезивања вредности и одређеног обележја конкретног објекта.

Две основне групе обележја су:

- категоријска (квалитативна) и
- нумеричка (квантитативна).

У категоријска обележја спадају номинална (енгл. nominal) и ординална (енгл. ordinal) обележја, док се у нумеричка обележја убрајају интервална (енгл. interval) и размерна (енгл. ratio) обележја.

Разлике између ове четири врсте обележја засноване су на различитим операцијама које су подржане над вредностима тих врста. Номинална обележја дозвољавају поређење типа једнако или различито (= и ≠), док код вредности ординалних обележја постоји и поредак, тј. дозвољене су и друге операције поређења (<, ≤, ≥ и >). Над интервалним обележјима су, поред претходних шест операција, могуће и операције сабирања и одузимања, док се над размерним обележјима, уз све већ наведене операције, могу обављати и операције множења и дељења [58].

У Табели 1, дат је пример једног скупа података који је дефинисан над пет обележја и садржи три појаве.

Табела 1: Пример скупа података, табеле

Име	Пол	Датум рођења	Стручна спрема	Плата
Вукан	М	01.01.2001.	Висока	1000
Милутин	М	02.02.2002.	Виша	900
Јевросима	Ж	03.03.2003.	Висока	950

Три обележја су категоријална: Име и Пол су номинална, док је Стручна спрема ординално обележје. Преостала два обележја су нумеричка: Датум рођења је интервално, а Плата је размерно обележје.

3.2. Проблемски домен

Главни предмет истраживања је проналажење методолошког приступа за смештање инстанци CIM модела у релациону базу података, омогућавање праћења промена над инстанцама модела, као и рестаурирање инстанци у неко од претходних стања.

Смештање CIM инстанци у релациону базу података захтева озбиљно истраживање, јер просто коришћење готових алата који пресликавају CIM објекте у шему базе података (ORM), може резултовати неоптималним решењем. Конкретно, како CIM представља један комплексан модел, сами подаци су распоређени по апстрактним и конкретним класама. Пошто већина конкретних класа зависи од већег или мањег броја апстрактних класа, тако је већина општих података дефинисана кроз апстрактне класе. Како једна инстанца модела садржи само конкретне класе и ако се свака класа пресликава директно на један скуп података (случај, табелу) у бази података, следи да би један упит за читање једног скупа података, пресликаног на основу једне конкретне класе, морао да преузме и све остале скупове података који су пресликани на основу апстрактних класа, а од којих зависи циљана конкретна класа. Исти проблем постоји и уколико бисмо желели да постојећу појаву неке конкретне класе података изменимо или избришемо или да додамо нову.

Праћење промена над инстанцама CIM модела може бити корисно, јер постоји потреба да се бележе извршене промене. При томе, није важно обезбедити увек чување комплетних инстанци модела у датом тренутку, него само појединачне измене које су настале. Како се измене над CIM инстанцама извршавају у групама, ради ефикасне размене измена над моделом као што је објашњено у одељку 3.1, сличан принцип могао би бити искоришћен и за праћење промена. Потребно је истражити могућност да измене извршене над елементима инстанци модела буду аутоматски меморисане, како за поједине елементе, тако и по везама које постоје у групи измена.

Рестаурирање инстанци CIM модела битно је уколико је потребно омогућити враћање инстанци модела на претходно стање. Да би рестаурирање било ефикасно, потребно је пронаћи минимални скуп елемената који би морали бити враћени у жељено претходно стање.

3.3. Могући начини меморисања инстанци CIM модела

Сврха CIM модела је да дефинише интерфејс између програма које користе ENTSO-E чланови, како би могли да размењују информације о стањима електроенергетске мреже. ENTSO-E од свих системских оператора (TSO), који су чланови асоцијације, захтева размену података о стањима електроенергетске мреже за коју су задужени.

Комплетне инстанце CIM модела, као што је описано у поглављу 3.1, по спецификацији [19] записане су у XML датотекама, или, скраћено, у CIM/XML формату. Информације о променама било ког дела инстанце CIM модела, унутар ENSTO-E асоцијације, одвија се разменом датотека у сажетом CIM/XML формату. Да би се једна CIM/XML датотека сматрала валидном, она мора да задовољи основне захтеве [19]:

- CIM/XML датотека мора да задовољи XML спецификацију, верзија 1.0 (http://www.w3.org/TR/REC-xml).
- CIM/XML датотека мора да подржи низ правила дефинисаних кроз Упрошћену RDF синтаксу за електроенергетске системе, као што је дефинисано кроз IEC 61970-552 спецификацију.
- CIM/XML датотека мора да садржи CIM елементе који су валидни према CIM RDF дефиницији шеме (XSD).

Из наведеног могло би се закључити да је просто снимање инстанци CIM модела у систему датотека (енгл. File System) сасвим довољно и поуздано решење. Међутим, у односу на карактеристике DBMS-ова, системи датотека имају неколико озбиљних мана:

- контролише се само физички приступ датотекама, али не и унутрашња логичка структура датотека,
- не постоји начин да се на вишем нивоу апстракције искажу упити који би добавили све жељене податке који су записани у датотекама,
- приступ једној датотеци обично је оптимизован само за једног корисника,
- нема контроле редудантности,
- не постоји предефинисан и поуздан начин да се направи сигурносна копија података и да се они рестаурирају и
- подаци нису обавезно изоловани од других, небитних, података.

Због наведених разлога, чување инстанци CIM модела у систему датотека искључује се из даљег разматрања.

Како CIM/XML датотека мора да садржи CIM елементе који су валидни према CIM RDF дефиницији шеме, тада CIM/XML представља и имплементацију Ресурсно орјентисаног оквира (RDF). Одатле би могло следити да је снимање инстанци CIM модела у RDF базу података прави и једини избор.

Аутори Шулте, Бербнер, Стеинметз и Услар у свом раду [56] анализирају стратегије за смештање CIM инстанци. Један од начина да се формира RDF шема базе података је путем већ постојећег RDF/XML пресликавања. Као последица свега тога чини се да је коришћење RDF база података, које припадају групи NoSQL база података, прави избор

за чување инстанци CIM модела. Међутим, аутори подсећају и да CIM стандард не нуди ни један званични начин за чување самих модела. Из тог разлога, аутори анализирају предности и мане коришћења релативно нових RDF база података, у односу на већ афирмисане и свеприсутне релационе базе података. Са друге стране, аутори Шумилин, Стуки, Син и Хагенмејер у [9] не објашњавају зашто су изабрали да користе RDF базе података за чување инстанци CIM модела.

Како се наводи у [56], главни аргумент за коришћење RDF база података је да су веома дистрибуиране, не захтевају шему базе података и доста су скалабилне. Главни недостатак коришћења релационих база података, како је закључено у [56], јесте да се сви подаци морају унапред пресликати са CIM-XML формата на шему базе података. Међутим, главни разлог због којег аутори у [56] предлажу коришћење релационих база података јесу временске перформансе.

На Графику 1 приказан је однос временских перформанси када се користи приступ снимања CIM инстанци у релациону базу података у односу на приступ снимања CIM инстанци у RDF базу података. На хоризонталној оси приказани су скупови података, где први скуп има 11 елемената, други 472 елемента, а трећи, последњи, има 23.352 елемента. На вертикалној оси приказано је време потребно да се одређени скуп упише у базу података.

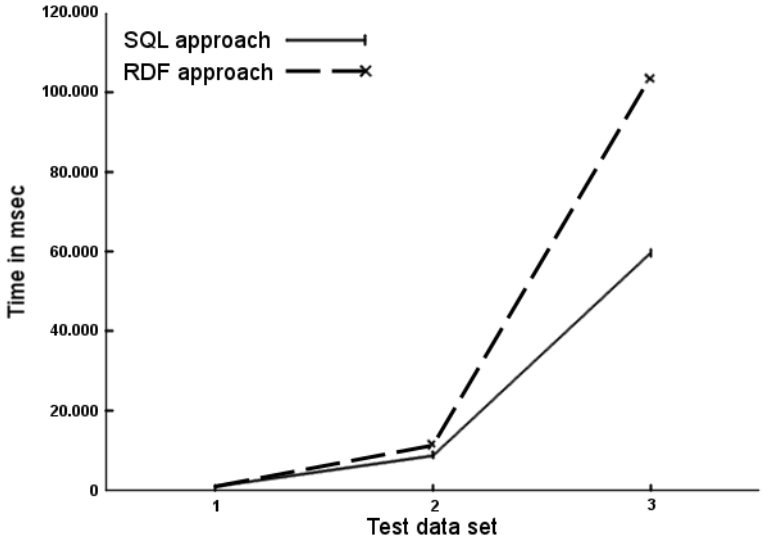


График 1: Писање CIM инстанци у базу података

Исти скупови података коришћени су и за читање CIM инстанци из базе података и снимање у датотеку, односно у CIM/XML формат. На Графику 2 приказан је однос временских перформанси када се користи приступ читања CIM инстанци из релационе базе података у односу на приступ читања CIM инстанци из RDF базе података.

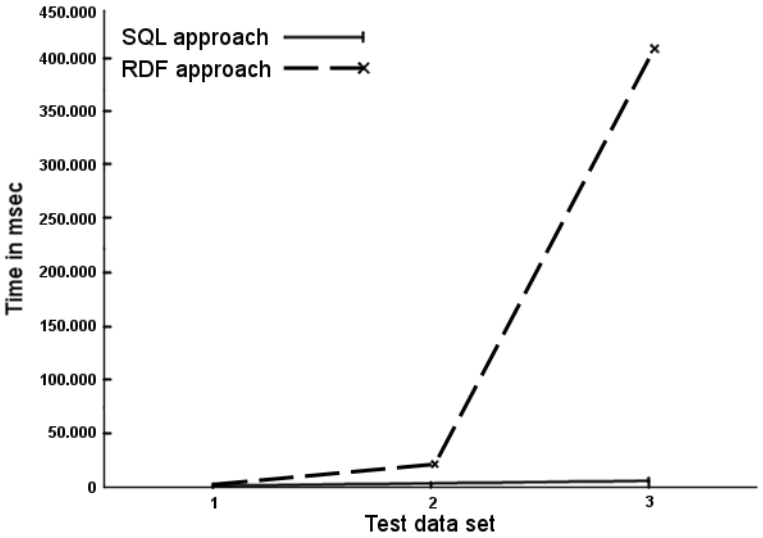


График 2: Читање CIM инстанци из базе података и снимање у датотеку

На основу дијаграма приказаних на графицима 1 и 2, јасно се види да је време потребно за ове операције дуже за RDF базе података у односу на релационе базе података. Ово важи и за читање и за писање CIM инстанци.

Аутори Равикумар, Капарде и Прадееп у [22] уочавају како се многа софтверска решења суочавају са проблемима при размени информација са стањима инстанци CIM модела. Као главни проблем, аутори издвајају недостатак DBMS који је пројектован управо за рад са инстанцама CIM модела. Односно, уочава се недостатак CIM орјентисане базе података (CIMODB). За реализацију такве базе података аутори оспоравају коришћење NoSQL база података, првенствено због свеприсутности релационих база података.

Такође, поједини аутори, Леавит у [46], као и Најак, Порија и Пооџари у [6] скрећу пажњу и на потенцијално високу комплексност NoSQL база података. Аутори наводе упитну поузданост, слабу конзистентност и стандардизацију, недовољно познавање конкретне технологије, као и ограничен број алата за рад са NoSQL базама података као главне препреке при коришћењу тих база података.

Аутори Кумар и Прабу у [42] пореде NoSQL базе података и релационе базе података. Они уочавају неконзистентност при коришћењу NoSQL база података, мањак стандарда и означавају их као да су још увек у развоју. Ради поређења временских перформанси NoSQL и релационих база података, аутори бирају Cassandra базу података и MySQL базу података, респективно. На основу извршених провера, аутори уочавају да је NoSQL база података знатно бржа од релационе базе података при упису података. Међутим, релациона база података бржа је при читању података. На основу овога, може се закључити да је један тип базе података бољи од другог у зависности од случаја коришћења. Слично уочавају и аутори Лидиа, Шанкар, Памина и Раја у [18] који уочавају да постоје велике варијације у перформансама NoSQL база података и да је за право поређење потребно бележити и потом анализирати перформансе NoSQL база података у реалним условима коришћења.

3.4. Закључак поглавља

У овом поглављу представљени су основни појмови који обухватају опис CIM модела, његову унутрашњу структуру и начин размене података о променама извршеним над инстанцом модела.

Описан је и проблемски домен, односно, проналажење методолошког приступа за смештање инстанци CIM модела у релациону базу података, омогућавање праћења промена над инстанцама модела, као и рестаурирање инстанци у неко од претходних стања.

Могући начини меморисања инстанци CIM модела детаљно су анализирани. Као што је наведено у одељцима 2.3 и 3.3, основни начин чувања инстанци CIM модела је у систему датотека, као CIM/XML датотека. Међутим, тај приступ чувања инстанци одбацује се, јер се контролише само физички приступ датотекама, а не постоји начин да се поставе упити, нема контроле редудантности, сигурност података је ограничена и нема изолације података.

RDF, као и цела групација NoSQL база података, такође се не разматрају због слабих временских перформанси, високе комплексности, упитне поузданости, слабе конзистентности и стандардизације, као из због недовољног познавања конкретне технологије.

На основу анализе у поглављима 2.3 и 3.3, у овом раду предлаже се коришћење релационе базе података. Главни аргументи за такву одлуку су у бољим или бар условно бољим временским перформансама, њиховој свеприсутности у информационим технологијама, високом нивоу поузданости, конзистентности и поштовању утврђених стандарда.

4. Приступ развоју апликативног решења за управљање инстанцама CIM модела

Развој методолошког приступа за управљање инстанцама CIM модела није једноставан процес. Због комплексности CIM модела који је у основи методолошког приступа и недефинисаног начина чувања у самим инстанцама, не постоји готово и јединствено решење за управљање инстанцама CIM модела. Поред тога, постоји потреба да се омогући брз и релативно једноставан приступ тренутним, активним стањима CIM инстанци. Такође, очекује се приступ и историјским стањима поменутих инстанци, као и могућност да се оне рестаурирају у неко од претходних стања.

Како би све наведено било омогућено и јасно образложено, потребно је развити методолошки приступ који ће по фазама и корацима дати решење за управљање инстанцама CIM модела.

4.1. Структура методолошког приступа

Методолошки приступ за управљање инстанцама CIM модела подељен је у три основне фазе, од којих свака садржи одређене кораке:

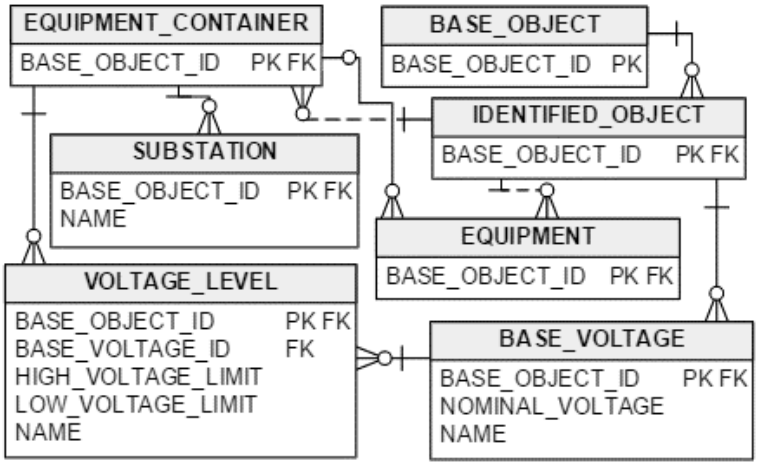
- Фаза 1: Развој шеме базе података за смештање инстанци CIM модела и управљање инстанцама CIM модела.
  - Корак 1: Анализа активног модела и његово поређење са CIM моделом.
  - Корак 2: Дефинисање структуре активног модела.
- Фаза 2: Развој шеме базе података за праћење историје инстанци CIM модела.
  - Корак 1: Дефинисање структуре за евидентирање промена над једним елементом.
  - Корак 2: Дефинисање структуре за праћење промена над групом елемената.
  - Корак 3: Дефинисање структуре за рестаурацију инстанце модела.
- Фаза 3: Развој поступка за рестаурацију инстанци CIM модела.
  - Корак 1: Дефинисање алгоритма за поништавање појединачне измене.

Наведене фазе и кораци биће детаљно описани у следећим поглављима.

4.2. Ф1: Развој шеме базе података за смештање инстанци CIM модела и управљање инстанцама CIM модела

У овом поглављу биће описана прва фаза методолошког приступа. У овој фази је потребно дефинисати структуру шеме базе података за смештање инстанци CIM модела. Таква шема базе података треба да омогући смештање тренутних стања инстанци модела и зато се за њу уводи појам *активни модел*.

Активни модел представља репрезентацију CIM модела, у којој свака класа CIM модела има своју представу у активном моделу. Пошто је парадигма релационих база података за чување инстанци модела одабрана, активни модел је шема базе података имлементирана за релационе DBMS. На основу тога, табеле базе података, дефинисане активним моделом, називамо *активном базом*. Пошто се у активној бази смештају инстанце CIM модела, те инстанце се могу називати и *активне инстанце* CIM модела. На Слици 2 дат је исечак шеме базе активног модела, коју ћемо детаљно разрадити у наставку поглавља. Ради бољег увида у комплексност шеме базе активног модела, у Прилогу 2 је дат нешто шири исечак модела у односу на Слику 2.



Слика 2: Исечак шеме активног модела

У овој фази методолошког приступа основни циљ је да се дефинише начин смештања CIM инстанци у релациону базу података, као и да се омогући брз и релативно лак приступ ка њима. Како би се постигао тај циљ, ова фаза је подељена у два корака:

- Корак 1: Анализа активног модела и његово поређење са CIM моделом.
- Корак 2: Дефинисање структуре активног модела.

У наставку овог поглавља детаљно ће бити приказани наведени кораци. Брзо добављање тренутног стања модела изузетно је важно за електроенергетске прорачуне као што су:

- прорачун токова снага (енгл. Load Flow – LF),
- анализа сигурности (енгл. Contingency Analysis – CA),
- прорачун преносних капацитета (енгл. Net Transfer Capacity – NTC) и
- прорачун кварова (енгл. Fault Calculation - FC).

4.2.1. Ф1-К1: Анализа активног модела и његово поређење са CIM моделом

У овом кораку методолошког приступа биће специфициран активни модел и поступак његовог креирања. Као што је већ казано, на Слици 2 дат је исечак циљног активног модела. Како је CIM дефинисан кроз апстрактне и конкретне класе, ради креирања активног модела, апстрактне класе CIM модела су пресликане на тзв. *апстрактне табеле* шеме базе. По истом принципу, конкретне класе су пресликане на тзв. *конкретне табеле* активног модела.

Како по CIM спецификацији не постоји једна класа која дефинише јединствени идентификатор и коју наслеђују све остале класе, овде је уведена BaseObject класа, која није део CIM спецификације. Такође, у активном моделу је уведена BASE_OBJECT табела, која је добијена пресликавањем BaseObject класе. BASE_OBJECT табела је уведена како би све остале табеле, које је реферишу директно или посредно, имале страни кључ, који је добијен на основу њеног примарног кључа (BASE_OBJECT_ID). У овом приступу наслеђивање између класа пресликава се на референциране табеле које су добијене пресликавањем родитељске класе, тако што се у табели која референцира и која је добијена пресликавањем класе наследнице креира страни кључ, који је истовремено и примарни кључ.

У циљу додатне провере оправданости увођења предложеног методолошког приступа, потребно је анализирати два типична приступа за добијање модела који су концептуално слични активном моделу. Пошто је CIM специфициран путем UML језика користећи Enterprise Architect алат, исти алат би се могао искористити за генерисање шеме базе која би одговарала активном моделу. Такође, како се класе добијене генерисањем из CIM UML модела могу пресликати на шему базе, исти приступ се може применити и за добијање активног модела. За те потребе, може се искористити просто објектно-релационо пресликавање (ORM). Постоји много општих алата за потребе објектно-релационог пресликавања. Међутим, код комплексних модела као што је CIM, са много хијерархијских нивоа између класа, потенцијални проблем је релативно велик број апстрактних класа [53]. Уколико се свака класа просто преслика на одговарајућу табелу, за читање података из било које табеле било би тешко написати једноставан упит. Како би се добавили подаци из *родитељских* табела које су добијене пресликавањем родитељских класа, неопходно је коришћење JOIN клаузула. Што се више JOIN клаузула користи у упитима, то је спорије њихово извршавање. С друге стране, циљ је да се омогући формирање једноставних упита који се релативно брзо извршавају.

4.2.2. Ф1-К2: Дефинисање структуре активног модела

У овом кораку прве фазе, у циљу омогућавања писања релативно једноставних и брзих упита, потребно је креирати ORM на специфичан начин. Прво је потребно пресликати атрибуте класа на колоне њихових табеларних представника, али само за конкретне класе. У овом случају, у пресликавање су укључени и атрибути наслеђени од свих класа предака.

Потом, табеле које представљају апстрактне табеле креирају се само са неопходним примарним кључевима, без икаквих других колона. У овом приступу, апстрактне

табеле су неопходне само због правилног постављања релација. На пример, као што је приказано на Слици 2, табели EQUIPMENT је неопходна релација ка табели EQUIPMENT_CONTAINER. Табела EQUIPMENT_CONTAINER добијена је пресликавањем апстрактне класе Equipment Container, а она се може инстанцирати само у облику својих наследника, односно или као Substation, или као Voltage Level. Дакле, релација ка табели EQUIPMENT_CONTAINER посредно представља релацију или ка табели SUBSTATION, или ка табели VOLTAGE_LEVEL.

Коначно, референце дефинисања у апстрактним, родитељским класама пресликавају се као страни кључеви у табелама добијеним пресликавањем конкретних класа, које наслеђују одговарајућу апстрактну класу.

Овако организована шема базе омогућује формирање једноставних упита над само једном табелом за добављање података једног типа података. Овим приступом избегнута је потреба за коришћењем JOIN клаузуле при добављању података. Практично, сви потребни подаци су, фигуративно говорећи, „гурнути доле“ ка конкретим класама, док апстрактне табеле служе као тзв. „костур“ модела.

4.2.3. Закључак фазе

Пројектујући активну базу на описан начин, омогућено је постављање упита ка бази без потребе да се користе JOIN клаузуле. На тај начин постиже се знатно брже добављање података о тренутном стању конкретне инстанце модела.

У циљу обезбеђења праћења промена над инстанцама модела, њиховог рестаурирања, анализа података, статистичке обраде инстанци, или потенцијалног предвиђања будућих промена, активни модел мора бити проширен. Таква проширења биће анализирана у следећој фази предложеног приступа.

4.3. Ф2: Развој шеме базе података за праћење историје инстанци CIM модела

У овој фази потребно је дефинисати структуру шеме базе података за смештање историјских података о променама над инстанцама CIM модела. Таква шема базе података представља проширење шеме базе податка за опис активног модела, те је у питању једна, обједињена шема базе података. Подела је логичка и има циљ да раздвоји описивање табела за смештање тренутних стања и табела за смештање историјских стања. Из тог разлога, за опис тзв. историјских табела уводи се појам *историјски* модел. Као у претходном одељку, табеле дефинисане историјским моделом називамо *историјском базом података*.

У овој фази методолошког приступа основни циљ је да се дефинише начин смештања промена над подацима које носе инстанце CIM модела. При томе, треба имати у виду и потребу да се добави инстанца модела у неком од претходних стања, односно да се обави њена рестаурација. У циљу постизања тог циља, ова фаза је подељена у 3 корака:

- Корак 1: Дефинисање структуре за праћење промена над једним елементом.
- Корак 2: Дефинисање структуре за праћење промена над групом елемената.
- Корак 3: Дефинисање структуре за рестаурацију инстанце модела.

У наставку овог одељка наведени кораци биће детаљно описани. Праћење историјских стања инстанци модела изузетно је важно за електроенергетске прорачуне као што су:

- анализа сигурности (енгл. Contingency Analysis – CA), за цео дан у склопу процедуре за прогнозу загушења током следећег дана (енгл. Day Ahead Congestion Forecast - DACF),
- прорачун преносних капацитета (енгл. Net Transfer Capacity – NTC), за цео дан у склопу DACF процедуре и
- статистика губитака у моделу за дефинисани период времена, којој се задаје датум од-до тако да је могуће радити и за целу годину.

Како се већина прорачуна извршава на нивоу дана или дуже, а гранулација података је за сваки сат, потребно је за сваки дан складиштити 24 инстанце модела за сваку мрежу од интереса. Уколико би се чувале 24 комплетне инстанце CIM модела који се врло мало разликују, очекује се да би таква база података захтевала доста меморијског простора. Зато је потребно пројектовати базу података која би на оптимизован начин пратила промене над подацима.

Овде је потребно нагласити да се може очекивати да ће најчешће измене над инстанцама CIM модела бити у оквиру његова три профила, SSH, SV и TP. То се очекује јер SSH, SV и TP пакети, како је описано у одељку 3.1 и CIM спецификацији [19], представљају стабилна стања, променљива стања и топологију мреже, односно, улазне податке, излазне податке и резултате процесирања мреже, респективно. Ниво гранулације, у односу на то колико често долази до неке промене у подацима, за профиле SSH, SV и TP обично је један сат. За неке друге пакете, као што је EQ пакет, ниво гранулације може бити знатно дужи, мерен месецима или дуже. У предложеном приступу тежња је да буде развијен историјски модел и касније поступак за рестаурацију инстанци, који ће радити подједнако добро, без обзира на ниво гранулације за поједине делове инстанци CIM модела.

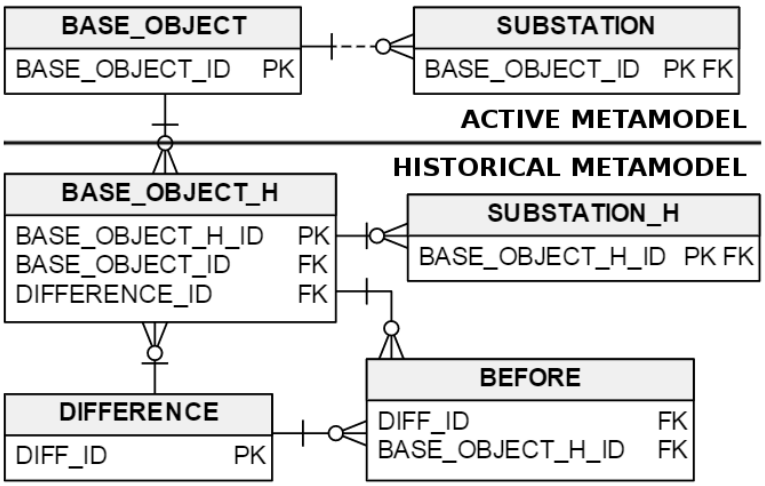
Такође, сваки од наведених прорачуна у одељку 4.2. који се врше над тренутним стањем инстанце модела, може се вршити и над инстанцом модела рестаурираном из историје.

4.3.1. Ф2-К1: Дефинисање структуре за праћење промена над једним елементом

У овом кораку методолошког приступа, биће предложен историјски модел за праћење промена над једним елементом. Под *елементом*, у овом тексту, подразумева се један ред у табели која је добијена пресликавањем класе на активни модел у релационој шеми базе. Један ред у табели представља пресликану једну инстанцу класе која припада опису CIM модела.

На Слици 3 дат је исечак предложеног историјског модела. За праћење промена над једним типом елемента, односно, променама над једним редом из конкретне табеле активног модела, потребно је креирати једну *историјску* табелу. Свака конкретна табела добија своју историјску табелу која ће бити коришћена за смештање претходног стања сваког елемента. У исечку на Слици 3, за табелу која представља Substation класу, креирана је табела SUBSTATION_H. Све историјске табеле имају примарни кључ који је истовремено и страни кључ, а добијен на основу примарног кључа из BASE_OBJECT_H

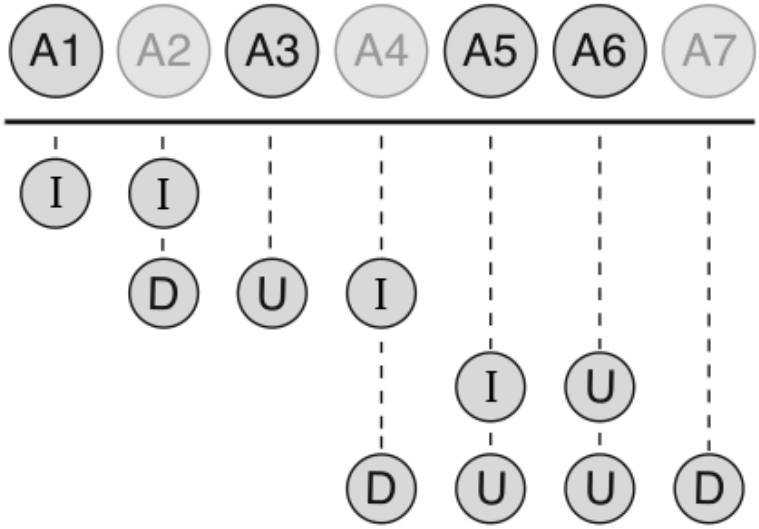
табеле. BASE_OBJECT_H табела је добијена пресликавањем BaseObject History класе и има страни кључ ка BASE_OBJECT табели из активног модела. На овај начин успостављена је веза између активног и историјског модела. Ради бољег увида у комплексност шеме базе активног модела, у Прилогу 3 је дат нешто шири исечак модела у односу на Слику 3.



Слика 3: Исечак шеме историјског модела

BaseObject History класа, односно њена пресликана табела, носи информацију о томе која акција је извршена, акција додавања, модификације или брисања (енгл. Insert, Update, Delete). Пре него што се било која акција изврши над елементом активне базе, његово тренутно стање евидентира се у одговарајућој табели историјске базе података. За те записе уводи се појам *историјски елемент*. На тај начин, сваки елемент из активне базе може добити своју историју промена у историјској бази. За праћење историје једног елемента уводи се појам *низ историјских елемената*, који обухвата све промене извршене над једним елементом.

На Слици 4, дата је илустрација односа елемената из активне базе података и њихових историјских елемената у историјској бази података. Елементи из активне базе података представљени су круговима изнад хоризонталне линије, од A1 до A7. Њихова претходна стања, односно, њихови историјски елементи, такође су приказани круговима, али су позиционирани вертикално испод њих, испод хоризонталне линије. Слова I, U и D представљају извршену акцију: додавање, модификација и брисање (енгл. Inser, Update, Delete), респективно. Низови историјских елемената представљени су испрекиданим линијама, које се пружају вертикално испод елемената активне базе података и повезују их са њиховим историјским елементима. Елементи активне базе података, који су приказани засенчено, у датом тренутку више не постоје у активној бази података, јер је последња активност извршена над њима била брисање.



Слика 4: Историјски елементи

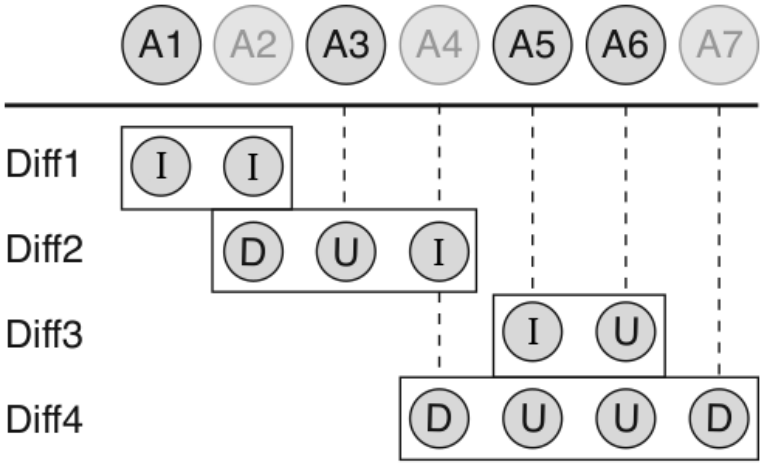
Креирајући историјске табеле остварена је могућност да се прате промене над једним елементом активне базе. Међутим, рестаурација комплетне инстанце модела на стање у одређеном тренутку може бити веома захтевна. Користећи тренутно описану шему базе, за рестаурацију било би потребно проћи кроз све историјске елементе и поредити их по задатом датуму и времену.

4.3.2. Ф2-К2: Дефинисање структуре за праћење промена над групом елемената

У овом кораку методолошког приступа биће проширена шема историјског модела за груписање измена над инстанцом модела које су се одиграле у једној итерацији. За такву структуру уводи се појам *група разлика* (енгл. Difference Group – DG) и представља датотеку разлика (енгл. Difference file – Diff), дефинисану у одељку 3.1.

Како је приказано на Слици 3, група разлика у историјском моделу представљена је табелом DIFFERENCE. Историјске табеле повезане су са DIFFERENCE табелом путем BASE_OBJECT_H табеле. Сваки ред у DIFFERENCE табели представља једну групу разлика. Практично, историјски елементи настали у једној итерацији, груписани су у групу разлика.

На Слици 5, дата је илустрација односа историјских елемената и група разлика. Групе разлика представљене су правоугаоницима који обухватају историјске елементе настале у једној итерацији. Групе разлика именоване су називима Diff1 до Diff4.



Слика 5: Групе разлика

Груписањем историјских елемената лакше је пронаћи измене које су настале симултано, у склопу једне итерације. Креирањем једне групе разлика, уједно се креира и *контролна тачка* (енгл. Checkpoint) у којој је могућа рестаурација дате инстанце модела. Битно је нагласити и да, када се инстанца модела рестаурира по контролним тачкама, додатно је обезбеђено да ће инстанца модела остати у валидном стању. На пример, уколико се из инстанце модела уклони један елемент, потребно је уклонити и све елементе који од њега зависе да би инстанца остала валидна. Практично, да би се поништила једна група измена, нема потребе да се траже сви неопходни историјски елементи поређењем времена њиховог настанка.

Међутим, да би инстанца модела била рестаурирана релативно једноставно и брзо, није довољно да постоје историјски елементи и да су груписани у групе разлика. Пратећи Слику 5, може се уочити да, уколико постоји само једна група разлика (Diff1), једноставно је рестаурирати инстанцу модела у претходно стање. Додати елементи треба да се уклоне, модификованим елементима да се врате претходне вредности, а избрисани елементи треба да се додају назад у инстанцу модела.

Рестаурација је нешто сложенија уколико постоји дужи низ измена над инстанцом, на пример, уколико би постојале само две групе разлика, Diff1 и Diff2. Да би инстанца била рестаурирана у стање пре Diff1, прво би требало поништити измене наведене у Diff1. Потом, треба поништити измене наведене у Diff2, али без преклапајућих елемената, што је у датом случају елемент A2. Практично, уочава се да је и даље потребно поредити потенцијално велику количину података да би инстанца модела била рестаурирана. Што више група разлика постоји, то ће требати више времена да се инстанца модела рестаурира у жељено стање.

4.3.3. Ф2-К3: Дефинисање структуре за рестаурацију инстанце модела

У овом кораку методолошког приступа, поново ће бити извршена проширења историјске шеме базе података, сада за потребе рестаурирања модела. За те потребе уводе се ознаке *први* и *последњи* (енгл. is_first, is_last, респективно). Ознаке *први* и *последњи* представљају две нове колоне, типа тачно или нетачно, у табели BASE_OBJECT_HISTORY. На овај начин, за сваки елемент из активне базе података једноставно је утврдити који од његових историјских елемената је први, а који је

последњи. При томе, ознака *последњи* мора се модификовати сваки пут када се забележи нова промена над елементом. Прва промена над елементом има ознаку *први*, и то се не мења докле год постоји тај запис. Ако постоје историјски елементи, тачно један је први и тачно један је последњи.

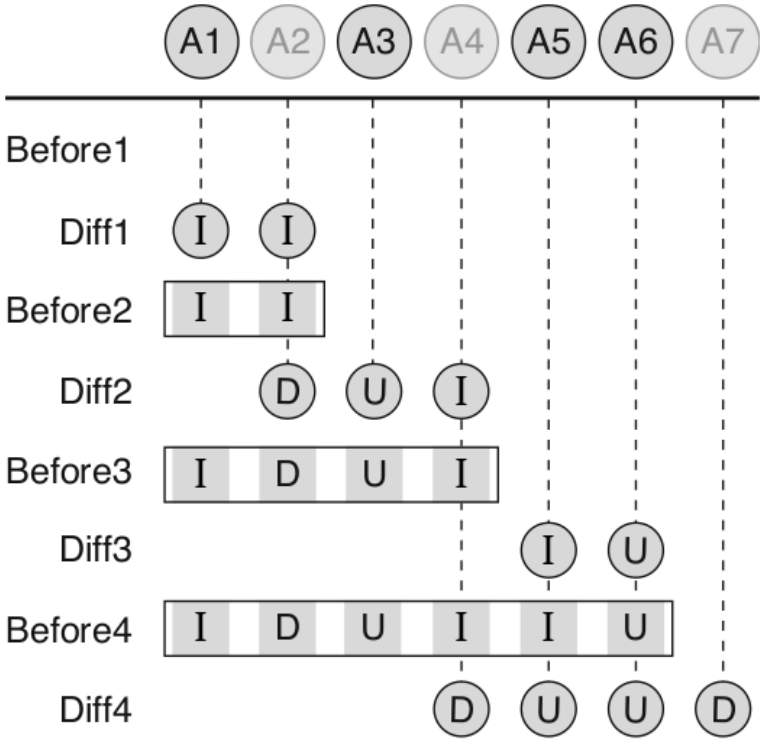
Потом је потребно креирати структуру која ће груписати све претходне последње измене које постоје у историјској бази података. За такву структуру уводи се појам *претходне разлике* (енгл. Before Group of Changes). На Слици 3 претходне разлике представљене су табелом BEFORE. Како би табела претходних разлика била ажурна, пре извршења било којих измена над активном и историјском базом података, потребно је све постојеће историјске елементе са ознаком *последњи* забележити у претходним разликама. Свака претходна промена (BEFORE табела) везује се за групу разлика (Difference табела) које се потом примењују. Када се примене нове измене над инстанцом модела, ознака *последњи* ажурира се за сваки низ историјских елемената. Свака група разлика има своје претходне разлике.

На Слици 6 приказане су претходне разлике заједно са групама разлика којима припадају. Претходне разлике, означене са Before1 до Before4, представљене су правоугаоницима који обухватају историјске елементе, док су историјски елементи представљени квадратима. За групу разлика Diff3, претходна разлика је Before3, која обухвата историјски елемент од A1, који је додат кроз Diff1 и историјске елементе за A2, A3 и A4, који су избрисани, модификовани и додати кроз Diff2, респективно.

Уочава се на примеру на Слици 6 да постоје историјски елементи који нису обухваћени претходним разликама. На пример, измене над елементима A5, A6 и A7 настале су након креирања Before3 кроз промене које су груписане у Diff3 и Diff4. Како би инстанца модела била рестаурирана у стање пре Diff3, измене настале након креирања Before3 такође се морају поништити. Из тог разлога, уводи се појам *наредне разлике* (енгл. After Group of Changes).

Наредне разлике састоје се од историјских елемената који су настали након претходних разлика. За дефинисање наредних разлика није потребно увођење посебне структуре која би их обухватила. Гледано из угла претходних разлика, да би неки елемент био уврштен у групу наредних разлика, мора да испуни два услова:

1) да има ознаку *први* у свом низу историјских промена и
2) да се ниједан историјски елемент из његовог низа промена не налази у датој групи претходних разлика.

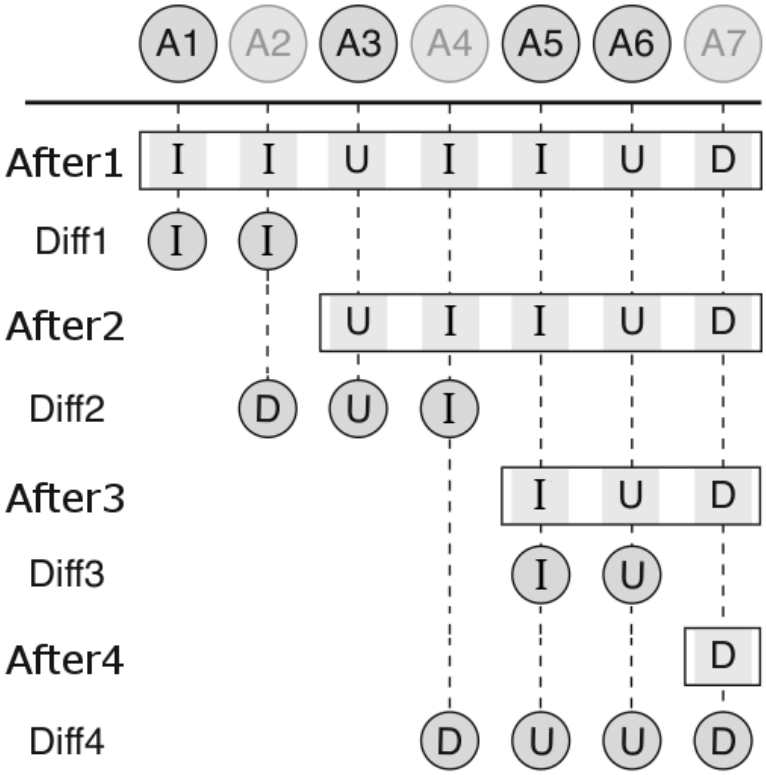


Слика 6: Претходне разлике

На Слици 7, приказане су наредне разлике заједно са групама разлика којима припадају. Наредне разлике, означене са After1 до After4, представљене су правоугаоницима који обухватају историјске елементе, док су елементи представљени квадратима. За групу разлика Diff3 наредна разлика је After3 која обухвата историјске елементе A5 и A6, који су додати и модификовани кроз Diff3, респективно, и историјски елемент за A7, који је избрисан кроз Diff4. Елемент A4 који је избрисан у Diff4, претходно је додат у Diff2, и та активност је забележена у Before3. Из тог разлога, нема потребе да се неки историјски елемент за A4 нађе у групи наредних измена. На датом примеру види се и да After3 обухвата све измене начињене кроз Diff3. Та ситуација је валидна, јер су наредне разлике логичка структура дефинисана на основу табеле претходних разлика.

Користећи групу *претходних разлика* и групу *наредних разлика* могуће је растаурирати инстанцу модела. Са овако дефинисаном шемом базе, за рестаурацију инстанце модела нема потребе да се претражује историја промена поредећи времена када су измене настале.

Са овако дефинисаном историјском базом података, са групама разлика – *претходних разлика* и *наредних разлика*, постоји солидна основа да се рестаурација инстанце модела обави на високо оптимизован начин.



Слика 7: Наредне разлике

4.3.4. Закључак фазе

У овом поглављу описана је друга фаза методолошког приступа. За потребе праћења промена над инстанцама CIM модела шема базе података проширена је историјским моделом. За праћење промена над једним елементом активне базе уведени су историјски елементи повезани у низ историјских елемената. Како би рестаурација модела могла бити извршена без потребе да се историјске промене траже по времену када су настале, уведене су групе разлика, претходних разлика и наредних разлика.

Историјски модел, како је дефинисан у овом одељку, треба да омогући рестаурирање модела са што мање упита над базом података. Конкретан поступак рестаурације инстанце модела биће објашњен у следећем поглављу.

4.4. Ф3: Развој поступка за рестаурацију инстанци CIM модела

У овој фази потребно је дефинисати конкретан поступак рестаурације инстанце CIM модела. Поступак рестаурације обавља се кроз један корак и ослоњен је на активну и историјску базу података и у потпуности искоришћава структуре које су дефинисане у претходном одељку. За потребе дефинисања поступка за рестаурацију инстанци CIM модела уводи се појам *пресек разлика* (енгл. Slice). Пресек разлика није дефинисан посебном структуром, него је сачињен од:

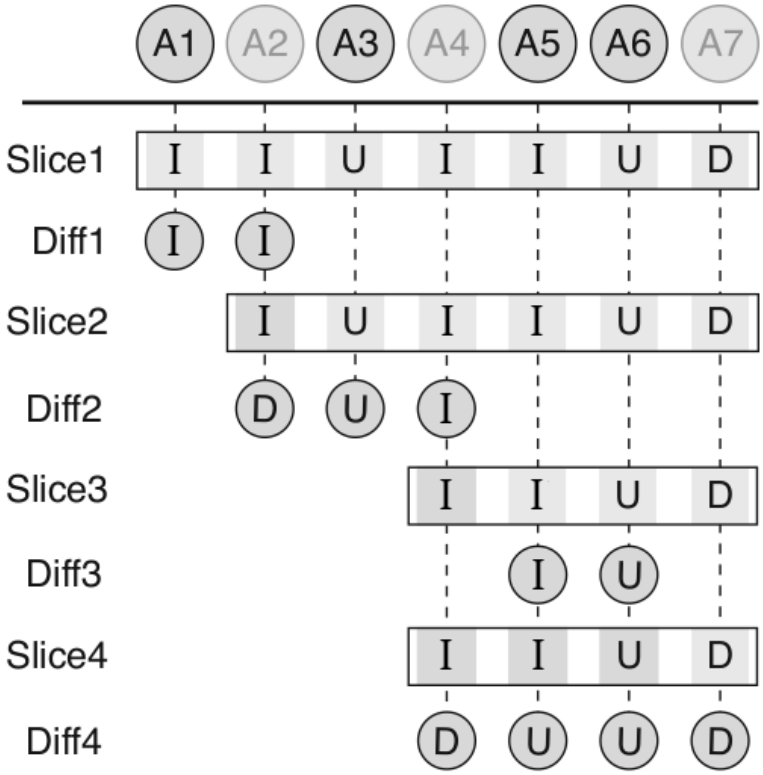
- претходне разлике, али без историјских елемената који су са ознаком *последњи* и
- наредне разлике, у односу на дату претходну разлику.

Историјски елементи претходне разлике са ознаком *последњи* не улазе у састав пресека разлика јер описују промене које су тренутне у активном моделу и не треба их поништавати.

Овако дефинисан пресек разлика обухвата све историјске елементе који су потребни да би инстанца CIM модела била рестаурирана у жељеној контролној тачки. Свака група разлика представља једну контролну тачку. Како група разлика има своје групе претходних и наредних разлика, самим тим има и свој пресек разлика. На тај начин, инстанца модела може бити рестаурирана у стања дефинисана контролним тачкама.

4.4.1. Ф3-К1: Дефинисање алгоритма за поништавање појединачне измене

У овом кораку методолошког приступа, ослањајући се на пресек разлика, биће дефинисан алгоритам за поништавање појединачне измене. На Слици 8, приказани су пресеци разлика заједно са групама разлика којима припадају. Пресеци разлика означени са Slice1 до Slice4, представљени су правоугаоницима који реферишу историјске елементе, док су елементи представљени квадратима. За групу разлика Diff3, пресек разлика је Slice3, који обухвата историјски елемент од A4, који је додат кроз Diff2; историјске елементе за A5 и A6, који су додати и модификовани кроз Diff3; и коначно историјски елемент за A7, који је обрисан кроз Diff4. На датом примеру лако се може уочити да је за рестаурирање инстанце модела у првобитно, иницијално стање, потребно највише података. Међутим, такође се може уочити да је мање података потребно како би модел био рестауриран у стања која су скорије настала.

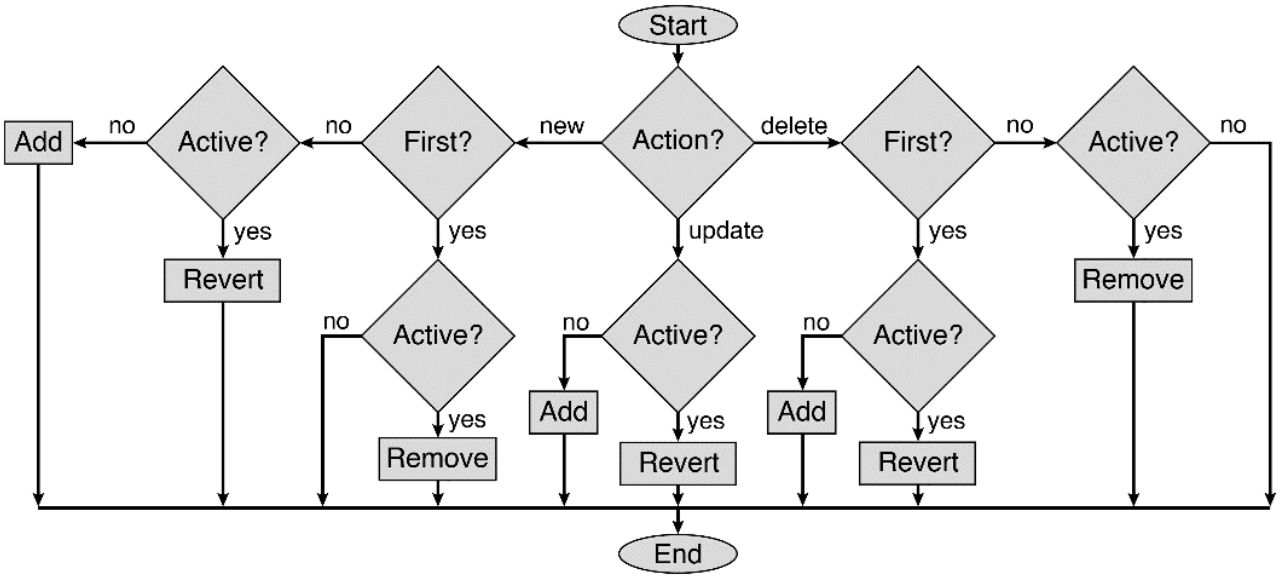


Слика 8: Пресек разлика

За извршење поступка рестаурације инстанце CIM модела потребно је добавити из базе података активни модел и пресек разлика који одговарају контролној тачки у коју

желимо да вратимо инстанцу модела. Пресек разлика садржи историјске елементе који описују појединачне акције које су примењене над активним моделом.

Алгоритам за поништавање појединачне измене итерира по пресеку разлика. Зависно од акције историјског елемента, ако има ознаку *први* и ако постоји у активном моделу, у алгоритму се примењује одговарајућа операција како би инстанца модела била враћена у пређашње стање. Дијаграм тока алгоритма дат је на Слици 9, а табеларни приказ алгоритма је дат у Табели 2. Следећи Слику 9 и Табелу 2, јасно се види зависност извршених акција и операција за њихово поништавање.



Слика 9: Дијаграм тока алгоритма за поништавање појединачне измене

У Табели 2, тачком „•“ означено је стање историјског елемента. Симболима „⊤“ и „⊥“ означена је тачност или нетачност тврдњи, а симболом „✓“ селектује се операција коју треба извршити у циљу поништења описане промене.

Из Слике 9 и Табеле 2 произилази, ако је историјски елемент забележио акцију „Додато“, постоје четири условне операције за поништавање:

- ако историјски елемент има ознаку *први* и његов елемент постоји у инстанци активног модела, извршава се акција „Удаљити“ елемент из инстанце модела, јер је елемент био додат након контролне тачке;
- ако историјски елемент има ознаку *први*, а његов елемент не постоји у инстанци активног модела, извршава се акција „Ништа“, јер је елемент био додат и избрисан након контролне тачке;
- ако историјски елемент нема ознаку *први* и његов елемент постоји у инстанци активног модела, извршава се акција „Вратити претходне“ вредности елементу из инстанце модела, јер је елемент био додат пре контролне тачке и модификован након контролне тачке; и
- ако историјски елемент нема ознаку *први*, а његов елемент не постоји у инстанци активног модела, извршава се акција „Додати“ елемент назад у инстанцу модела, јер је елемент био додат пре контролне тачке и избрисан након контролне тачке.

Табела 2: Алгоритам за поништавање појединачне измене

Извршена акција			Има ознаку први?	Постоји у активној инстанци?	Операција за поништавање			
Додато	Модификовано	Обрисано			Додати	Вратити претходно	Удаљити	Ништа
●			⊤	⊤			✓	
●			⊤	⊥				✓
●			⊥	⊤		✓		
●			⊥	⊥	✓			
	●		⊤ / ⊥	⊤		✓		
	●		⊤ / ⊥	⊥	✓			
		●	⊤	⊤		✓		
		●	⊤	⊥	✓			
		●	⊥	⊤			✓	
		●	⊥	⊥				✓

Из Слике 9 и Табеле 2 произилази, ако је историјски елемент забележио акцију „Модификовано“, његова ознака да ли је први није од важности. У овој ситуацији постоје две условне операције за поништавање:

- ако елемент постоји у инстанци активног модела, извршава се акција „Вратити претходне“ вредности елементу из инстанце модела, јер је елемент био модификован или пре, или након контролне тачке; и
- ако елемент не постоји у инстанци активног модела, извршава се акција „Додати“ елемент назад у инстанцу модела, јер је елемент био модификован или пре, или након контролне тачке, а потом избрисан након контролне тачке.

Из Слике 9 и Табеле 2 произилази, ако је историјски елемент забележио акцију „Обрисано“, постоје четири условне операције за поништавање:

- ако историјски елемент има ознаку *први* и његов елемент постоји у инстанци активног модела, извршава се акција „Вратити претходне“ вредности елементу из инстанце модела, јер је елемент прво био избрисан након контролне тачке, а потом додат са истим кључем;
- ако историјски елемент има ознаку *први*, а његов елемент не постоји у инстанци активног модела, извршава се акција „Додати“ елемент назад у инстанцу модела, јер је елемент избрисан након контролне тачке;

- ако историјски елемент нема ознаку *први* и његов елемент постоји у инстанци активног модела, извршава се акција „Удаљити“ елемент из инстанце модела, јер је елемент био избрисан пре контролне тачке и додат након контролне тачке; и
- ако историјски елемент нема ознаку *први*, а његов елемент не постоји у инстанци активног модела, извршава се акција „Ништа“, јер је елемент био избрисан пре контролне тачке.

4.4.2. Закључак фазе

Овако дефинисан поступак за рестаурацију инстанци CIM модела омогућује потпуну рестаурацију у било којој контролној тачки. За рестаурацију потребно је из базе података добавити тренутну, активну инстанцу модела и пресек разлика. Комплетан поступак рестаурације инстанце одвија се у само једној итерацији, проласком кроз пресек разлика и примењујући алгоритам за поништавање појединачне измене.

4.5. Закључак поглавља

У овом поглављу представљен је оригинални методолошки приступ за управљање инстанцама CIM модела. Методолошки приступ је због своје комплексности, али и због комплексности самог CIM модела, представљен у неколико фаза.

У првој фази описан је активни модел као део шеме базе кроз који се дефинише како се инстанце CIM модела снимају у базу података. На основу тога, табеле дефинисане кроз активни модел називамо активном базом података. У овој фази дата је анализа односа класа које дефинишу сам CIM модел и табела које су добијене њиховим пресликавањем на шему базе података. Активни модел је пројектован тако да омогући писање што једноставнијих упита ка бази података и да се тиме постигну боље временске перформансе.

У другој фази је описан историјски модел као део шеме базе података, кроз који се дефинише како се бележи историја промена која се извршила над инстанцама CIM модела снимљеним у активној бази података. На основу тога, табеле дефинисане кроз историјски модел називамо историјском базом. У овом делу описана је структура за праћење промена извршених над једним елементом CIM инстанце из активне базе података. Потом, дат је опис структуре за груписање промена у групе разлика, која уједно служи и као контролна тачка за каснију рестаурацију модела. У следећем кораку дат је опис структуре претходних разлика за праћење промена које су се десиле пре примене дате групе разлика. У склопу ове фазе описана је и група наредних разлика за коју није потребно дефинисање посебне структуре.

У последњој фази описан је поступак за рестаурацију инстанци CIM модела. У овом делу описан је и пресек разлика који садржи све историјске елементе потребне да би инстанца CIM модела била враћена у стање пре одабране контролне тачке. Као део поступка за рестаурацију дат је и детаљан опис алгоритма за поништавање појединачне измене из пресека разлика.

У овом приступу развијени историјски модел и поступак за рестаурацију инстанци не разликују одвојене пакете CIM модела. Њихове структуре и функционалности

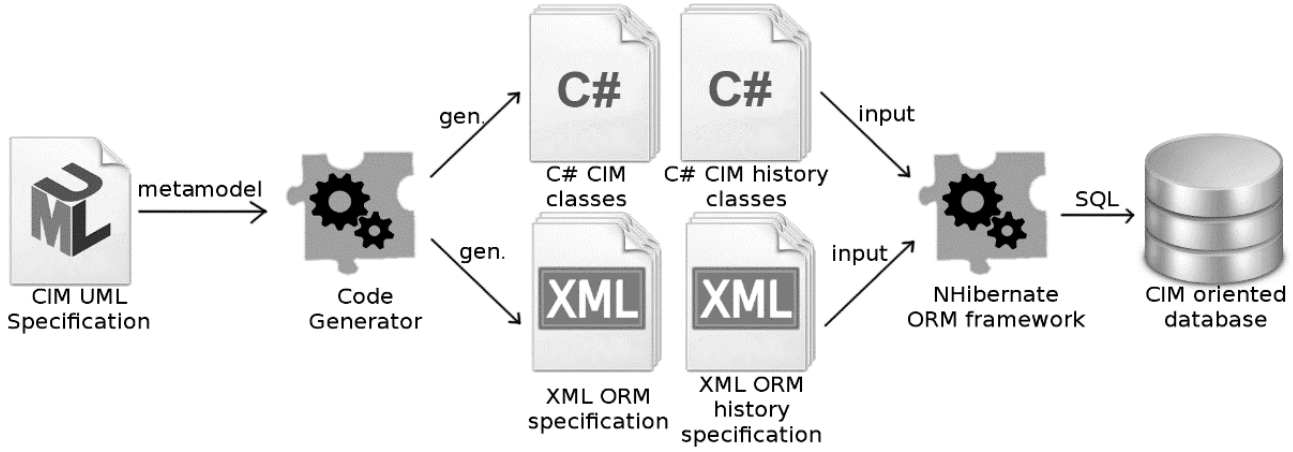
заснивају се на класама које дефинишу CIM модел, односно, на табелама које су добијене објектно-релационим пресликавањем. Због нивоа грануларности, очекује се да ће највише историјских елемената бити повезано са елементима чије класе су дефинисане у SSH и SV пакетима. Међутим, пошто сви елементи имају исти третман, то ни на који начин не утиче на поступак праћења историје и рестаурације инстанце.

Према досадашњим сазнањима аутора, развијени приступ апликативног решења за управљање инстанцама CIM модела представља нов методолошки приступ. Овим се доприноси унапређењу поступка пројектовања шеме базе података за смештање инстанци модела, праћења измена над инстанцом модела и рестаурацији инстанце модела у жељено стање.

5. Развој софтверске апликације за меморисање инстанци CIM модела и управљање инстанцама CIM модела

У претходном поглављу дат је детаљан опис оригиналног методолошког приступа развоју апликативног решења за управљање инстанцама CIM модела. У том приступу предвиђено је формирање табела путем објектно-релационог пресликавања, али на специфичан начин. Такође, за сваку конкретну табелу предвиђа се одговарајућа историјска табела. Ради груписања измена уведена је табела за групу разлика, а ради праћења историје уведена је табела за претходне разлике.

Имајући у виду да је CIM модел описан са око 600 међусобно повезаних класа, од којих је око 200 конкретних класа, може се уочити да би имплементација описаног приступа била доста захтевна. Из тог разлога, по узору на радове представљене у одељцима 2.4. и 2.5, предлаже се увођење Генератора кода. Развојни процес кроз који се имплементира методолошки приступ приказан је на Слици 10.



Слика 10: Развојни процес

У наставку овог поглавља детаљно ће бити описани: Генератор кода који је развијен за потребе развоја циљне софтверске апликације, затим делови циљне апликације који се добијају као производи самог Генератора, шема базе са активним и историјским моделом, угњеждене процедуре за смештање инстанци CIM модела, функције за управљање инстанцама CIM модела, као и реализација поступка за рестаурацију инстанци CIM модела.

5.1. Пројектовање и реализација Генератора кода за потребе развоја циљне софтверске апликације

У овом раду, Генератор кода можемо сматрати централном компонентом развијеном за потребе развоја циљне софтверске апликације. Око 95% изворног кода који чини циљну апликацију, добијено је као производ рада Генератора кода. Управо делови софтверске апликације који изискују највише напора за развој, шема базе, угњеждене процедуре за смештање инстанци CIM модела, програмске функције за управљање инстанцама CIM модела, реализовани су генерисањем њиховом изворног кода.

Улазни податак за Генератор кода је CIM UML спецификација, конкретно модели класа. На основу тога, Генератор кода зна које класе чине CIM и који су њихови међусобни односи. Пошто је за циљно окружење одабрана .NET технологија [43], Генератор кода, на основу CIM спецификације, генерише C# класе. Оне служе за учитавање и снимање инстанце CIM модела и за пресликавање на шему базе по објектно-релационом принципу. Након тога, генеришу се C# класе за праћење историје, тзв. *историјске класе*. Оне служе за смештање података о историји промена и за објектно-релационо пресликавање на шему базе. Инстанце историјских класа представљају историјске елементе у бази података.

За потребе овог истраживања, Генератор кода је наменски писан и чине га:

- Генераторски метамодел, део за опис CIM модела,
- Генератор класа, део за креирање објектно орјентисаног CIM модела,
- Генератор шеме, део за креирање објектно-релационог пресликавања,
- Генератор процедура, део за креирање угњеждених процедура и
- Генератор функција, део за креирање програмских функција.

За све делове Генератора кода, осим Генераторског метамодела, опционо је позивање њихових функционалности. Ово је омогућено јер није увек пожељно изнова креирати све делове циљне софтверске апликације који се добијају генерисањем изворног кода. Неретко се дешава да је потребно дорадити делове генерисаног кода како би он био валидан.

Како би се на основу CIM UML спецификације произвела жељена функционалност, потребно је да ту спецификацију учита програм, у овом раду Генератор кода, који на основу ње производи потребни изворни код. Ради једноставнијег приступа опису CIM модела, Генератор кода учитава CIM UML спецификацију у посебан модел класа. Тиме се подиже ниво апстракције и добија се CIM метамодел. У овом раду, CIM метамодел је представљен Генераторским метамоделом и чине га класе које садрже описе класа, поља и релација којима је дефинисан CIM модел. При генерисању кода, прво се учита CIM UML спецификација у Генераторски метамодел и тек потом се наставља са даљим радом. У приказу развојног процеса, на Слици 10, учитавање спецификације означено је стрелицом са натписом „metamodel“.

Након што се CIM UML спецификација учита у Генераторски метамодел, Генератор класа може да креира класе које представљају објектно орјентисани CIM модел. За сваку класу из спецификације креира се одговарајућа класа циљног програмског

језика. При генерисању, поштује се дефинисано наслеђивање између класа, поља и релације. За свако поље и релацију се додатно генеришу одговарајући пропертији. Оваква функционалност могла се постићи и креирањем програмских класа користећи сам алат у којем је направљена спецификација. Међутим, када се наменски развија генератор кода за генерисање класа, релативно је једноставно у креиране класе додати произвољне функционалности као што су копи-конструктор, функција за клонирање дате инстанце класе, анотације за пресликавање у XML формат и слично. За потребе овог истраживања, Генератор класа додатно генерише и историјске класе, али само за класе које су у спецификацији означене како конкретне. Рад Генератора класа описан је детаљније у одељку 5.2. У приказу развојног процеса, на Слици 10, креирање објектно орјентисаног CIM модела означено је стрелицом са натписом „gen“ и упереном ка C# иконицама.

Следеће што Генератор кода може да уради је креирање XML спецификације кроз које ће бити дефинисано објектно-релационо пресликавање. За сваку генерисану класу генерише се по једна XML спецификација. Ово је неопходно, јер, као што је описано у одељцима 4.2. и 4.3, објектно-релационо пресликавање је потребно обавити на специфичан начин. XML спецификације за основни објектни CIM модел дефинишу активну базу података, а XML спецификације за историјске класе дефинишу историјску базу података. Генератор шеме је задужен за функционалности описане у овом пасусу, а описан је детаљније у одељку 5.2. У приказу развојног процеса, на Слици 10, креирање XML спецификација означено је стрелицом са натписом „gen“ и усмереном ка XML иконицама.

У следећем кораку Генератор кода може да креира угњеждене процедуре намењене за специфицирану базу података. За ову функционалност задужен је Генератор процедура који се такође ослања на CIM спецификацију. Рад компоненте Генератор процедура описан је детаљније у одељку 5.3. У приказу развојног процеса, на Слици 10, креирање угњеждених процедура није посебно означено.

Потом, Генератор кода може да креира програмске функције намењене за комуникацију са базом података и оне чине једну библиотеку коју ћемо овде називати *Комуникациони слој* (енгл. API Communication Layer). За ову функционалност задужен је Генератор функција и он се такође ослања на CIM спецификацију. Рад Генератора функција је описан детаљније у одељку 5.3. У приказу развојног процеса, на Слици 10, креирање програмских функција није посебно означено.

У сажетку, након што је креиран објектни модел и начин његовог објектно-релационог пресликавања је специфициран, добија се шема базе података на основу које се креирају табеле у бази података. На овај начин добија се CIM орјентисана база података са активном и историјском базом података које представљају њене интерне делове. За потребе објектно-релационог пресликавања користи се NHibernate библиотека [50]. Ради што ефикаснијег искоришћења могућности одабране базе података, креиране су угњеждене процедуре, за које се потом креирају и одговарајуће програмске функције које их позивају.

5.2. Пројектовање и реализација шеме базе података за подршку циљне софтверске апликације

Као што је већ објашњено у Поглављу 4, у овом раду шема базе података сачињена је из два основна дела, из активног и историјског модела. Подела на два дела је логичка. Табеле дефинисане активним моделом служе за смештање тренутног стања инстанце CIM модела и представљају активну базу података, док табеле дефинисане историјским моделом служе за смештање претходних стања инстанце CIM модела и представљају историјску базу података.

Први корак у реализацији шеме базе података је креирање објектно орјентисаног CIM модела. У одељку 4.2. описан је приступ развоју шеме базе података за смештање и управљање инстанцама CIM модела. За те потребе, класе које описују активни модел могу строго да следе дефинисану CIM UML спецификацију. Међутим, класе за опис историјског модела потребно је дефинисати испочетка. Оне нису покривене CIM спецификацијом. Из тог разлога, класе за опис оба модела, и активног и историјског, креирају се користећи Генератор кода, конкретно Генератор шеме. Као што је приказано на Слици 10, Генератор кода учитава CIM UML спецификацију и на основу тога генерише потребне класе. Додатна погодност при генерисању кода је релативно лако мењање генерисаног кода. Тако се генерисане класе проширују да све имају једног, заједничког претка и уводи се класа BaseObject. На тај начин, једноставније је обухватити све елементе инстанце CIM модела и касније увести ограничења у шему базе да сви елементи активног модела имају јединствен примарни кључ. По сличном принципу, за историјске класе уводи се BaseObjectDiff класа.

Следећи корак у реализацији шеме базе података је пресликавање објектно орјентисаног модела на шему базе. Одабрана библиотека за потребе објектно-релационог пресликавања је NHibernate [50]. Како би се пресликавање обавило на специфичан начин, као што је објашњено у Поглављу 4, потребно је креирати XML спецификације, кроз које се дефинише објектно-релационо пресликавање. Формат спецификација је дефинисан путем XSD шеме, који је специфичан за NHibernate библиотеку. Због великог броја класа за које се мора написати начин пресликавања и специфичности самог пресликавања, и овде се прибегава коришћењу Генератора кода. Слично као у претходном кораку, као што се види на Слици 10, Генератор кода учитава CIM UML спецификацију и на основу тога генерише потребне XML спецификације за потребе пресликавања. Као пример XML спецификације за пресликавање дати су Прилог 4 и Прилог 5, где се види пресликавање класа BaseObject и BaseObjectDiff, респективно.

5.3. Реализација поступака за смештање инстанци CIM модела у реализовану базу података и уградња у циљну софтверску апликацију

Након што је реализована шема базе података, потребно је реализовати и поступак за смештање инстанци CIM модела у реализовану базу података и уградити га у циљну софтверску апликацију. За те потребе, очекује се постојање угњеждених процедура у бази података, као и функција које их позивају и чине Комуникациони слој.

Угњеждене процедуре обухватају скупове инструкција које се често користе. Поред тога, оне се брже извршавају од простих SQL наредби, јер се процес превођења наредби врши унутар базе података. Пошто је одабрана Oracle база података, угњеждене процедуре реализују се у PL/SQL језику (енгл. Procedural Language/Structured Query Language – PL/SQL). Он представља процедурално проширење SQL језика. За сваку класу из CIM модела, односно, за њену табелу добијену објектно-релационим пресликавањем, креиране су следеће процедуре:

- процедура за додавање елемента у табелу,
- процедура за додавање елемента у табелу с претходним додавањем потребних кључева у тзв. родитељске табеле,
- процедура за додавање елемента у табелу и у родитељске табеле уз бележење те активности додавањем историјског елемента и реферисањем одговарајуће контролне тачке,
- процедура за модификацију елемента у табели,
- процедура за модификацију елемента у табели уз бележење те активности додавањем историјског елемента и реферисањем одговарајуће контролне тачке,
- процедура за брисање елемента из табеле,
- процедура за брисање елемента из табеле с претходним брисањем потребних кључева у тзв. родитељским табелама,
- процедура за брисање елемента из табеле уз бележење те активности додавањем историјског елемента и реферисањем одговарајуће контролне тачке,
- процедура за читање елемената из табеле који припадају датој инстанци модела и
- процедура за читање пресека разлика за дату контролну тачку.

Пошто се инстанце CIM модела мењају у итерацијама од више измена заједно како би инстанца модела остала у валидном стању, једна група измена извршава се у оквиру једне трансакције. Уколико се евидентира историја измена, на почетку трансакције креира се контролна тачка, односно, група разлика за коју ће се везивати историјски елементи. Након тога, за нову контролну тачку креира се листа претходних разлика, и тек након тога се почиње са применом нових измена. У оквиру процедура које бележе историју води се рачуна и о освежавању ознака *први* и *последњи*. Креирање ових процедура треба да омогући релативно брз и једноставан приступ бази података. На

овај начин, сложеност шеме базе података остаје скривена за развојне инжењере који развијају друге делове апликације.

Како би се бази података приступило још једноставније, реализован је Комуникациони слој са функцијама које позивају описане угњеждене процедуре. Комуникациони слој је библиотека која омогућује објектни приступ бази података и њеним процедурама. Реализацијом ове библиотеке, уградња у циљну софтверску апликацију не би требало да буде компликована. На овај начин, уградња подршке за рад са CIM орјентисаном базом података у неку постојећу или нову програмску апликацију треба да буде знатно олакшана.

За развој Комуникационог слоја и угњеждених процедура базе података поново је коришћен Генератор кода. За креирање Комуникационог слоја задужен је Генератор функција, а за креирање угњеждених процедура задужен је Генератор процедура.

5.4. Реализација поступака за управљање инстанцама CIM модела и уградња у циљну софтверску апликацију

Након што је реализована шема базе података и поступак за смештање инстанци CIM модела у реализовану базу података, потребно је реализовати и поступак за управљање инстанцама CIM модела. За те потребе, потребно је правилно искористити функције развијене у претходном поглављу.

Како би рад са инстанцама CIM модела био што једноставнији, функције, развијене у претходном поглављу, групишу се у општије функције као што су:

- функција за додавање инстанце CIM модела у базу података,
- функција за добављање инстанце CIM модела из базе података,
- функција за измену инстанце CIM модела са листом елемената где је дошло до промена, са праћењем или без праћења историје,
- функција за добављање инстанци CIM модела из базе података и њено рестаурирање у жељеној контролној тачки и
- функција за брисање инстанце CIM модела из базе података.

Функција за додавање инстанце CIM модела у базу података подразумева иницијално снимање инстанце CIM модела. У поступку нема потребе да се за сваки елемент бележи још један историјски елемент који би говорио да је нешто додато. Одатле следи да се подаци додају само у активну базу.

Функција за добављање инстанце CIM модела из базе података подразумева читање тренутног стања инстанце CIM модела. Одатле следи да се подаци читају само из активне базе података. Као што је објашњено у одељку 4.2, ова функционалност би требало да буде релативно брза.

Функција за измену инстанце CIM модела са листом елемената где је дошло до промена без праћења историје подразумева измене само над подацима у активној бази података. Оваква могућност је подржана, како би системски оператери имали већу

слободу у свом раду. Очекује се да ће постојати потреба да се направе мање измене над активним моделом, за које неће бити потребе да се прави контролна тачка.

Функција за измену инстанце CIM модела са листом елемената код којих је дошло до промена са праћењем историје подразумева измене над подацима у активној и историјској бази података. Ово је препоручени начин управљања инстанцама CIM модела. Пре него што се направи било каква измена, креира се контролна тачка, као што је објашњено у одељку 4.3. Након тога, потребно је применити измене позивајући функције дефинисане у одељку 5.3.

Функција за добављање инстанце CIM модела из базе података и њено рестаурирање у жељеној контролној тачки подразумева читање тренутног стања инстанце CIM модела, као и пресек разлика. Активна инстанца CIM модела чита се из активне базе података, а пресек разлика из историјске базе података. Потом се инстанца рестаурира по поступку описаном у одељку 4.4.

Функција за брисање инстанце CIM модела из базе података подразумева брисање података о инстанци прво из историјске, а потом и из активне базе података.

За развој ових функционалности није потребно користити Генератор кода. Оцењује се да је напор који би се уложио у тзв. „ручно“ писање функција, мањи него када би се за то користио Генератор кода. Функције за управљање инстанцама CIM модела такође су део Комуникационог слоја.

5.5. Реализација поступка за рестаурацију инстанци CIM модела и уградња у циљну софтверску апликацију

Реализација поступка за рестаурацију инстанци CIM модела ослања се на претходно описане функционалности. Шема базе података, како за активни модел, тако и за историјски, моделована је тако да омогући поступак рестаурације инстанци CIM модела. Као што је описано у одељку 4.4, за рестаурацију су неопходни подаци из активне и историјске базе података.

Током рада са инстанцом CIM модела очекује се да оператер направи више измена. Очекује се да ће неки елементи бити додати, други избрисани, а многима ће се мењати вредности којима су описани. Тај поступак се одвија у итерацијама. У једној итерацији очекује се да ће доћи до измена више елемената. Зато се измена сваког појединачног елемента бележи, а измене настале у једној итерацији групишу се у једну групу разлика. Уведене су посебне стуктуре претходних и наредних разлика како би се лакше пратиле измене, као и сама рестаурација инстанце. За потребе рестаурације модела свака група разлика представља једну контролну тачку у коју инстанца модела може да се врати, односно, да се рестаурира.

Предуслов за процес рестаурације је постојање бар једне контролне тачке. Након што се одреди контролна тачка, из базе података добавља се активна инстанца CIM модела и пресек разлика који одговара контролној тачки. Сам поступак је детаљно описан у одељку 4.4. Рестаурација инстанце CIM модела извршава се тако што се поништавају измене које су забележене у пресеку разлика. Измене се поништавају једна по једна, примењујући алгоритам за поништавање појединачне измене, који је приказан на

Слици 9 и у Табели 2. Имплементација алгоритма за поништавање појединачне измене дата је у Прилогу 6. Ова функционалност такође је смештена унутар Комуникационог слоја.

5.6. Закључак поглавља

У овом поглављу представљен је развој софтверске апликације за меморисање инстанци CIM модела и управљање инстанцама CIM модела. Са својих неколико делова може се казати да развојни процес није једноставан.

Пројектовање и реализација шеме базе података за подршку циљне софтверске апликације, укључујући активни и историјски модел, овде су реализовани користећи аутоматско програмирање и развој вођен моделом. За те потребе је развијен Генератор кода.

Како би поступак за смештање инстанци CIM модела у реализовану базу података био реализован и како би била обезбеђена његова уградња у циљну софтверску апликацију, поново је коришћен Генератор кода. Генерисањем су имплементиране угњеждене процедуре унутар базе података, као и функције у Комуникационом слоју које их позивају.

За реализацију поступака за управљање инстанцама CIM модела, основне функције за рад са одвојеним типовима елемената CIM модела повезане су у неколико општих функција за рад са инстанцама CIM модела. Оне, такође, представљају Комуникациони слој.

При реализацији поступка за рестаурацију инстанци CIM модела ослањало се на све претходно урађено. Како би био омогућен поступак за рестаурацију, активни и историјски модел су моделовани на посебан начин и уведене су структуре како би се релативно лако и брзо налазили подаци потребни за рестаурирање. Поступак за рестаурацију заснива се на поништавању појединачних измена, а његов резултат је враћање целе инстанце CIM модела у жељено стање.

Развој софтверске апликације за меморисање и управљање инстанцама CIM модела великим делом зависи од резултата добијених генерисањем кода. Развојем Генератора кода добијен је програм који се користи за писање других програма. На тај начин значајно је смањено време потребно за развој и смањена је количина потенцијалних грешака у коду.

Сав програмски код за смештање, управљање и рестаурацију инстанци CIM модела представља Комуникациони слој. На овај начин, све операције потребне за рад са инстанцама CIM модела, смештене су у једну библиотеку. На тај начин, може се очекивати да би се уградња подршке за рад са инстанцама CIM модела у циљну софтверску апликацију свела на позивање ове библиотеке. При томе, циљна софтверска апликација може бити већ постојећа, као што су неки од TPA алата, или комплетно нова апликација.

6. Анализа и оцена развијеног приступа и софтверског решења

За потребе развоја апликативног решења за управљање инстанцама CIM модела развијен је нови методолошки приступ. У фазама и корацима дат је преглед како организовати шему базе за потребе активног и историјског модела, како омогућити праћење историје промена над инстанцом модела и, коначно, како приступити рестаурацији инстанци CIM модела.

Како би нови методолошки приступ био примењљив у пракси, користећи парадигму аутоматског програмирања и вођен развојем који је заснован на моделу, развијена је софтверска апликација за меморисање инстанци CIM модела и управљање инстанцама CIM модела. Дат је увид у поступак развоја шеме базе података, угњеждених процедура (енгл. Stored Procedure – SP) за базу података, као и програмских функција за комуникацију са таквом базом података.

У овом поглављу приказани су резултати анализе и оцене развијеног приступа и софтверског решења. Праћење историјских промена захтева проширење оригиналног модела и тиме се повећава комплексност решења. Када је поред извршења одређене акције потребно бележити и историју акција које су се догодиле, може се очекивати да ће доћи до пада временских перформанси. У наставку поглавља анализирано је неколико случајева у којима су временске перформансе од значаја. На сваком графику који следи у тексту, хоризонтална оса показује број елемената инстанце CIM модела који се тестира, а вертикална оса показује протекло време у секундама. Сви тестови изведени су на рачунару са процесором Intel Core 2 Duo E7500 2.93GHz и 8GB оперативне меморије. Коришћена база података (енгл. DBMS) је Oracle Database 11g Express Edition [52].

Због електроенергетских прорачуна наведених у одељку 4.2, очекује се да ће упити ка подацима смештеним у активну базу података бити доста чести. На Графику 3, приказано је просечно време потребно за добављање активне инстанце CIM модела. На графику се види упоредни приказ две функције за добављање инстанце:

- читањем из XML датотеке, што је приказано линијом са троугловима и
- читањем из релационе базе података, што је приказано линијом са круговима.

Може се уочити да време потребно за обављање функција линеарно зависи од броја елемената који чине инстанцу модела. Читање активне инстанце модела не зависи од количине историјских података. Као што се види у Табели 3, за читање инстанце CIM модела од 443 елемента потребно је 0,47 секунди, док је за читање инстанце од 237.540 елемента потребно 18,52 секунде. Такође, примећује се да је читање из XML датотеке спорије за мање инстанце CIM модела, док обрнуто важи за веће инстанце.

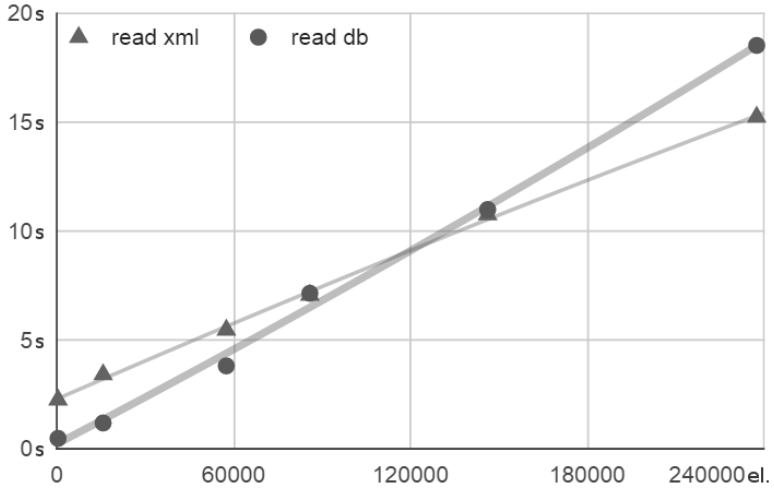


График 3: Читање инстанце CIM модела из XML-а и из базе података

Табела 3: Читање инстанце CIM модела из базе података

Број елемената	Потребно време [s]
443	0,47
15.743	1,17
57.548	3,81
85.898	7,14
146.243	10,98
237.540	18,52

Такође, неопходно је омогућити да се у базу података упише нова инстанца CIM модела. На Графику 4, приказано је просечно време потребно за уписивање активне инстанце CIM модела у базу података. На графику је видљив упоредни приказ две функције за упис нове инстанце:

- у XML датотеку, приказан линијом са троугловима и
- у релациону базу података, приказан линијом са круговима.

И овде се може уочити да време потребно за обављање функција линеарно зависи од броја елемената који чине инстанцу модела. Читање активне инстанце модела не зависи од количине историјских података. Као што се види у Табели 4, за упис инстанце CIM модела од 443 елемента потребно је 1,76 секунди, док је за упис инстанце од 237.540 елемента потребно 629,02 секунде. Такође, примећује се да је упис у XML датотеку знатно бржи у односу на упис у базу података. Ово се може објаснити чињеницом да је XML датотека бинарни запис који се прави серијализацијом података. С друге стране, CIM орјентисана база података, као што је описано у овом раду, има комплексну структуру шеме базе.

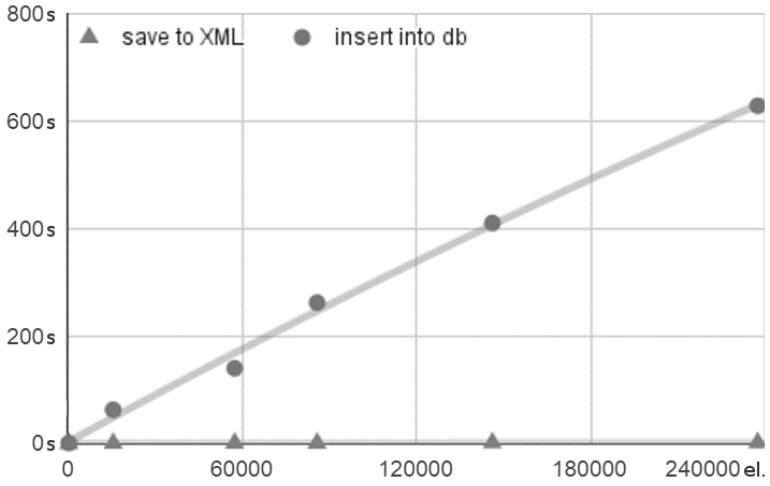


График 4: Снимање инстанце CIM модела у XML и у базу података

Табела 4: Упис инстанце CIM модела у базу података

Број елемената	Потребно време [s]
443	1,76
15.743	63,68
57.548	140,88
85.898	262,84
146.243	411,00
237.540	629,02

Због електроенергетских прорачуна наведених у 4.3, неопходно је омогућити мењање сачуване CIM инстанце у бази података, као и памћење направљених измена. На Графику 5 приказано је просечно време потребно за измену активне инстанце CIM модела. На графику се види упоредни приказ три функције за мењање инстанце:

- мењање инстанце CIM модела без праћења историје, приказано линијом са троугловима,
- мењање инстанце CIM модела са праћењем историје, приказано линијом са круговима и
- мењање инстанце CIM модела без праћења историје које је снимано у XML датотеку, приказано линијом са квадратима.

Пошто се при праћењу историје бележи свака промена над инстанцом CIM модела, за сваку операцију над CIM инстанцом извршава се бар још једна за бележење претходног стања. Као што се види у Табели 5, за измену 161 елемента, инстанце CIM модела од 443 елемента, потребно је 2,09 секунди, док је за измену 86.661 елемената, инстанце CIM модела од 237.540, потребно 737,27 секунде. На основу тога, може се закључити да је у просеку потребно 65% више времена за измену модела када се бележе извршене

операције. И овде се може уочити да време потребно за мењање инстанце линеарно зависи од броја елемената који чине инстанцу модела.

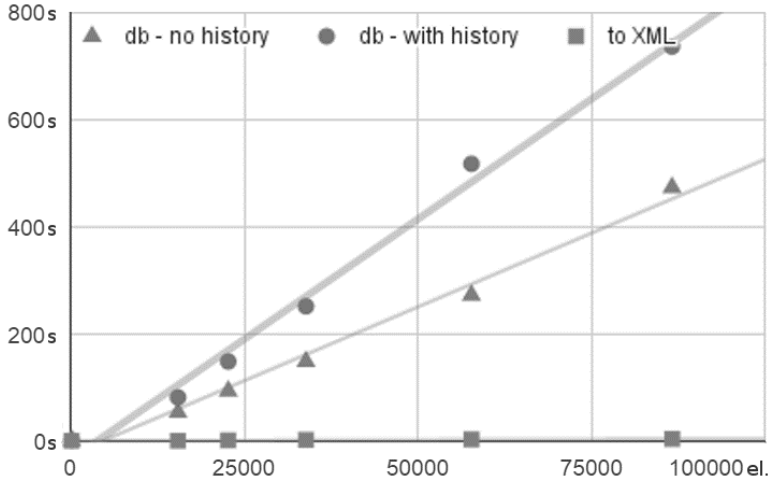


График 5: Измена инстанце CIM модела са праћењем и без праћења историје

Табела 5: Измена инстанце CIM модела са праћењем историје

Број елемената	Број промена	Потребно време [s]
443	161	2,09
15.743	15.480	82,48
57.548	22.721	149,70
85.898	33.921	252,17
146.243	57.761	518,09
237.540	86.661	737,27

Како би подаци за електроенергетске прорачуне који су наведени у 4.3 били добављени, неопходно је омогућити рестаурирање CIM инстанце у задатој контролној тачки. На Графику 6, приказано је просечно време потребно за читање активне инстанце CIM модела и њену рестаурацију. На графику се види упоредни приказ две функције:

- читање инстанце CIM модела из базе података, приказано линијом са троугловима и
- читање инстанце CIM модела из базе података са његовом рестаурацијом, приказано линијом са круговима.

За операцију рестаурирања инстанце CIM модела потребно је прво из базе података прочитати активну инстанцу. У следећем кораку добавља се *пресек разлика* (енгл. Slice) за жељену контролну тачку. На крају се пресек разлика примењује на активну инстанцу по поступку описаном у 4.4. Као што се види у Табели 6, за рестаурирање измена над 161 елементом, инстанце CIM модела од 443 елемента, потребно је 1,05

секунди, а за рестаурирање измена над 86.661 елементом, инстанце CIM модела од 237.540, потребно је 11,66 секунди. На основу тога, за дате примере може се закључити да је у просеку потребно 75% више времена да би инстанца модела била рестаурирана. Овде је важно приметити да је потребно мање времена за рестаурацију инстанце модела, него за њено преузимање из базе података.

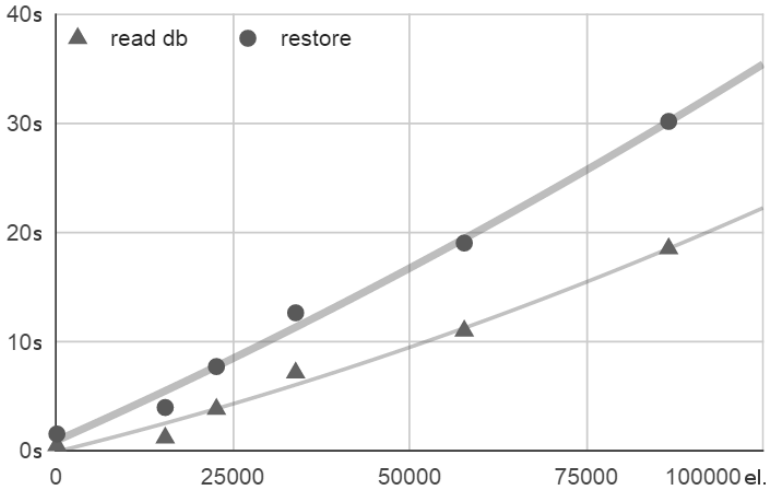


График 6: Читање инстанце CIM модела из базе података и његова рестаурација

Табела 6: Читање инстанце CIM модела из базе података и његова рестаурација

Број елемената	Број промена	Потребно време [s]
443	161	1,05
15.743	15.480	1,77
57.548	22.721	3,90
85.898	33.921	5,50
146.243	57.761	8,05
237.540	86.661	11,66

Поред анализе рада апликације, потребно је анализирати и ефикасност развоја софтверске апликације за меморисање инстанци CIM модела и управљање инстанцама CIM модела. Један од начина је поређењем броја линија кода (енгл. Lines of Code – LOC) добијених коришћењем Генератора кода у односу на број линија кода које су добијене простим развојем. У радовима који су наведени у [24] и [17], аутори наводе да је, редом, 75% и 48% изворног кода добијено генерисањем. При томе, мора се нагласити да су у питању алати опште намене. У овом раду, за потребе дефинисања шеме базе, угњеждених процедура базе података и *Комуникационог слоја*, око 95% изворног кода добијено је генерисањем. Овде се мора нагласити да је у питању наменски софтвер, без компоненти које нису дефинисане CIM моделом. Уколико би се ово решење укључило у већ постојећи софтвер, или се наставило са даљим развојем, може се очекивати да би се удео генерисаног кода постепено смањивао. Свакако, ово се може сматрати добрим

резултатом, јер су само одређени делови генерисаног кода захтевали интервенцију програмера. Програмер је писао и позивање генерисаних метода да би се остварила сложенија функционалност.

Додатна предност примене развоја вођеног моделом и аутоматског програмирања је брзо реаговање на промене. Промене могу настати у самом моделу, а могу бити и део развоја стратегије рада са инстанцама модела. Провера валидности архитектуре решења релативно брзо може бити спроведена над комплетним моделом, или над његовим већим делом који је од интереса. Закључује се да коришћење генератора кода знатно скраћује пут од идеје до реализације решења.

7. Закључак

У овом раду представљен је нови методолошки приступ развоју апликативног решења за управљање инстанцама CIM модела. У првој фази, описан је развој шеме базе података за смештање инстанци CIM модела и управљање инстанцама CIM модела. При томе је анализирана стуктура постојећег CIM модела и по узору на њу на специфичан начин моделован активни модел за смештање инстанци CIM модела. Циљеви развоја активног модела били су постизање релативно брзог приступа тренутним, активним стањима CIM инстанце. На основу тога можемо казати да је задовољена прва хипотеза:

> *X1: Постоји могућност да се инстанце CIM модела сместе у релациону базу података, тако да не дође до губитка података.*

У првој фази методолошког приступа описан је развој шеме базе података за праћење историје инстанци CIM модела. За те потребе развијен је историјски модел са:

- структуром за праћење промена над једним елементом инстанце CIM модела,
- структуром за груписање промена над инстанцом CIM модела које су настале у једној итерацији и
- структуром која ће омогућити рестаурацију инстанце CIM модела у стање пре задате контролне тачке.

На основу тога можемо казати да је задовољена друга хипотеза:

> *X2: Могуће је обезбедити праћење промена над инстанцама CIM модела смештених у релациону базу података.*

У последњој фази методолошког приступа описан је развој поступка за рестаурацију инстанци CIM модела. Развијени активни и историјски модел заједно чине једну шему базе података, на коју се поступак за рестаурацију у потпуности ослања. За опис алгоритма за рестаурацију једног елемента дат је дијаграм тока, као и његов табеларни приказ зависности извршене акције и операције за њено поништавање. На основу тога можемо казати да је задовољена трећа хипотеза:

> *X3: Могуће је обезбедити рестаурацију инстанци CIM модела из релационе базе података у неко од изабраних стања.*

Поред тога, описан је и развој софтверске апликације за меморисање инстанци CIM модела и управљање инстанцама CIM модела. Развијени Генератор кода олакшава реализацију шеме базе, поступака за смештање инстанци CIM модела, поступака за управљање инстанцама CIM модела и поступка за рестаурацију инстанци CIM модела. На тај начин, омогућено је релативно лако и ефикасно прилагођавање променама у CIM моделу. Додатно, применом парадигме аутоматског програмирања омогућено је

брже мењање и проширење саме апликације за рад са инстанцама CIM модела. На основу тога можемо казати да је задовољена и последња, четврта хипотеза:

> *X4: Развој генератора кода може да олакша развој CIM базе података и пратеће апликације.*

Коначно, дата је и анализа и оцена добијеног решења тако што су мерене временске перформансе за извршење операција снимања, читања, мењања и рестаурације инстанце CIM модела. Временске перформансе су приказане у графичком и табеларном облику, што даје бољи увид у понашање развијене апликације.

Решење описано у овом раду може послужити као добра основа за даљи развој CIM орјентисане базе података и програмске апликације која би се на њу ослонила. Даља истраживања за потребе рада са инстанцама CIM модела могу бити:

- развој подршке за складиштење података (енгл. Data Warehouse – DW) ради каснијег формирања извештаја и анализе података,
- развој подршке за предикцију (енгл. Forecasting Data – FD), како би на основу анализе забележених промена у историјској бази података била могућа предикција будућих стања система који се описује,
- проширење Генератора кода ради додавања подршке за генерисање осталих делова апликације, као што су компоненте за графички приказ, објављивање приступа путем сервиса и слично и
- проширење Генератора кода, ради пружања подршке за рад и са другим RDBMS.

Да би нека компанија имала успешну будућност, она мора да планира и доноси ваљане одлуке. Један од кључних услова, да би се правилно расуђивало, јесте узимање у обзир свих релевантних података. Складиштење података у DW требало би да послужи као добар извор података од интереса за пословање. Зато се DW може посматрати као систем за подршку при одлучивању.

CIM орјентисана база података, описана у овом раду, представља оперативну базу података (енгл. On-line Transaction Processing – OLTP). Основни циљ је омогућавање свакодневног пословања, а подаци се складиште на нивоу појединачних трансакција. Дубља анализа самих података једног таквог система је отежана и непогодна, јер је потребно анализирати и правити извештаје над оперативним подацима. Како оперативних података може бити изузетно много, њихова количина може експоненцијално да расте и зато их је неопходно архивирати. Говорећи метафорички, од дрвећа се не види шума.

С друге стране, мотивација за увођење система за подршку одлучивању је давање одговора на питања која занимају аналитичаре и управнике компанија. Општи принцип рада једног DW система подразумева:

- подаци се периодично извозе из операционе базе података,
- складиште се унутар посебног сервиса који представља DW базу података,
- при извозу, подаци се могу додатно чистити, форматирати, валидирати, организовати, сумирати и допуњавати подацима из других извора.

На овај начин, подаци се организују тако да одговарају потребама корисника система, односно, људима. Податке је потребно јасно повезати са другим подацима од интереса и везати их за јасно одређен временски период, не нарушавајући рад оперативне базе

података. Унутар DW подаци се могу организовати по темама: продаја, маркетинг, производња, пословање, транспорт и слично.

За DW, добро одређене димензије, шта се прати, и чињенице, вредности праћених параметара, од велике су важности. На основу тога, у зависности од временске грануларности, управа одређене компаније може добити одговоре на питања типа:

- Колика је производња одређене електране на дневном или месечном нивоу задате године?
- Колико је било оптерећење преносне електроенергетске мреже по сваком сату у задатом месецу и како се то мењало током године?
- Колика је просечна учесталост прекида напајања потрошача у задатом временском периоду?

Предикција будућих стања заснована на примени метода вештачке интелигенције (енгл. Artificial Intelligence – AI) обично подразумева два приступа: предикција помоћу Вештачких неуронских мрежа (енгл. Artificial Neural Network – ANN) или предикција помоћу Методе потпорних вектора (енгл. Support Vector Machine – SVM). Оба приступа захтевају постојање неког обучавајућег скупа података, који се у фази учења користи како би било могуће разумети систем података који је дат на анализу. Уколико је AI алгоритам правилно обучен, у оперативној фази могуће је до одређене мере предвидети будућа стања система. Историјска база података, описана у овом раду, складишти управо податке који су неопходни за фазу учења.

Предикцијом будућих стања могуће је одредити будуће трендове. Доношење пословних, финансијских и техничих одлука у многоме може бити олакшано уколико постоји ваљана предикција повећања или смањења потрошње електричне енергије, оптерећења мреже, броја корисника и слично.

У овом раду описани су само најнижи слојеви програмске апликације, база података и комуникациони слој. За развој осталих делова апликације, с обзиром на величину и комплексност CIM модела, може се очекивати да ће бити потребно уложити доста напора и времена. Како би развој компоненти као што су графички приказ, објављивање приступа путем сервиса и слично био олакшан, Генератор кода могао би се поново искористити додавањем подршке за генерисање наведених компоненти.

У компанијама које се баве развојем програмских апликација неретко долази до мењања одређених библиотека на које се одређени програм ослања. Исто се односи и на одлуку о промени базе података, односно RDBMS, које могу бити условљене техничким, финансијским или захтевима наручиоца. Код једноставнијих програмских апликација које се ослањају на неку од ORM библиотека, промена конкретног RDBMS обично би захтевала смо промену адресирања базе података. Међутим, у овом решењу база података је изузетно интегрисана у само програмско решење. Из тог разлога, промена базе података захтевала би и писање угњеждених процедура за нову базу података. У циљу избегавања додатног посла, поново може бити искоришћен Генератор кода. Проширење Генератора кода ради пружања подршке за рад и са другим RDBMS значајно би смањило напор потребан за прелазак на нову базу података.

Представљено решење намењено је управљању инстанцама CIM модела и њиховом складиштењу у базу података. Обухваћено је познавање структуре самог CIM модела,

парадигме аутоматског програмирања, база података и заједно обједињено у нови методолошки приступ. Даљи правци развоја зависиће од страна заинтересованих за коришћење и дистрибуцију овог решења.

Литература

[1] A. Armonas, L. Nemuraitė, „Pattern Based Generation of full-fledged Relational Schemas from UML/OCL Models“, Information Technology and Control, Vol. 35, No. 1, 2006, ISSN: 1392-124X.

[2] A. F. Case, „Computer-aided software engineering (CASE): technology for improving software development productivity“, ACM SIGMIS Database: the DATABASE for Advances in Information Systems, Vol. 17, No. 1, 1985, ISSN: 0095-0033, DOI: 10.1145/1040694.1040698.

[3] A. Fuggetta, „A classification of CASE technology“, Computer, Vol. 26, No. 12, pp. 25-38, 1993, ISSN: 0018-9162, DOI: 10.1109/2.247645.

[4] Á. Lédeczi, Á Bakay , M. Maróti, P. Völgyesi, G. Nordstrom, J. Sprinkle, G. Karsai, „Composing Domain-Specific Design Environments“, Computer, ISSN: 0018-9162, Vol. 34, No. 11, pp. 44-51, 2001, DOI: 10.1109/2.963443.

[5] A. M. Koss, „Programming on the Univac 1: a woman's account. Annals of the History of Computing“, IEEE, Vol 25, No. 1, 2003, ISSN: 1058-6180, DOI: 10.1109/MAHC.2003.1179879.

[6] A. Nayak, A. Poriya, D. Poojary, „Type of NOSQL Databases and its Comparison with Relational Databases“, International Journal of Applied Information Systems, 2013, ISSN: 2249-0868, DOI: 10.5120/ijais12-450888.

[7] A. Popović, I. Luković, V. Dimitrieski, V. Djukić, „A DSL for modeling application-specific functionalities of business applications“, Computer Languages, Systems & Structures (COMLAN), Elsevier, Vol. 43, pp. 69-95, 2015, ISSN: 1477-8424, DOI: 10.1016/j.cl.2015.03.003.

[8] A. R. Silva, „Model-driven engineering: A survey supported by the unified conceptual model“, Computer Languages, Systems & Structures, Vol. 43, pp. 139-155, 2015, DOI: 10.1016/j.cl.2015.06.001.

[9] A. Schumilin, K. U. Stucky, F. Sinn, V. Hagenmeyer, „Towards ontology-based network model management and data integration for smart grids“, Workshop on Modeling and Simulation of Cyber-Physical Energy Systems (MSCPES), ISBN: 978-1-5090-6437-3, 2017, DOI: 10.1109/MSCPES.2017.8064530.

[10] B. Wollenberg, J. P. Britton, E. Dobrowolski, R. Podmore, J. Resek, J. Scheidt, J. Russell, T. Saxton, C. Ivanov, „A Brief History: The Common Information Model [In My View]“, IEEE Power and Energy Magazine, Vol. 14, No. 1, 2016, ISSN: 1540-7977, DOI: 10.1109/MPE.2015.2481787.

[11] C. Atkinson, T. Kuhne, „Model-driven development: a metamodeling foundation“, IEEE Software, Vol. 20, No. 5, 2003, ISSN: 0740-7459, DOI: 10.1109/MS.2003.1231149.

[12] C. Henthorne, E. Tilevich, „Code Generation on Steroids: Enhancing COTS Code Generators via Generative Aspects“, Second International Workshop on Incorporating

COTS Software into Software Systems: Tools and Techniques (IWICSS '07), 2007, ISBN: 0-7695-2966-6, DOI: 10.1109/IWICSS.2007.4.

[13] C. Ivanov, „The Way to Exchange: What Is the Common Information Model? [Guest Editorial]“, IEEE Power and Energy Magazine, Vol. 14, No. 1, 2016, ISSN: 1540-7977, DOI: 10.1109/MPE.2015.2485902.

[14] C. Ivanov, D. Chury, „European electric power system on the way towards implementation of CIM based data exchange format“, Power & Energy Society General Meeting (PES '09), 2009, ISBN: 978-1-4244-4241-6, ISSN: 1932-5517, DOI: 10.1109/PES.2009.5275832.

[15] D. Akehurst, S. Kent, O. Patrascoiu, „A relational approach to defining and implementing transformations between metamodels“, Software & Systems Modeling, Vol. 2, No. 4, pp 215–239, 2003, ISSN: 1619-1366, DOI: 10.1007/s10270-003-0032-z.

[16] D. Capko, A. Erdeljan, S. Vukmirovic, I. Lendak, „A Hybrid Genetic Algorithm for Partitioning of Data Model in Distribution Management Systems“, Information Technology and Control, Vol. 40, No. 4, 2011, ISSN: 1392-124X, DOI: 10.5755/j01.itc.40.4.981.

[17] D. Kundu, D. Samanta, R. Mall, „Automatic code generation from unified modelling language sequence diagrams“, IET Software, Vol. 7, No. 1, 2013, ISSN: 1751-8806, DOI: 10.1049/iet-sen.2011.0080.

[18] E. L. Lydia, K. Shankar, P. Jeyakumar, B. Raja, „Correlating NoSQL Databases With a Relational Database: Performance and Space“, International Journal of Pure and Applied Mathematics, ISSN: 1311-8080, 117(7), Vol. 118, No. 7, pp. 235-244, 2018.

[19] ENTSO-E, Common Grid Model Exchange Standard (CGMES) – Based on IEC Common Information Model, version 2.4, 2014.

[20] G. Ravikumar, S. A. Khaparde, „A Common Information Model (CIM) Oriented Graph Database Framework for Power Systems“, DOI: 10.1109/TPWRS.2016.2631242.

[21] G. Ravikumar, S. A. Khaparde, R. K. Joshi, „Towards CIM implementation challenges in control centers worldwide and Indian power grid“, Annual IEEE India Conference (INDICON), 2014, ISBN: 978-1-4799-5364-6, ISSN: 2325-940X, DOI: 10.1109/INDICON.2014.7030580.

[22] G. Ravikumar, S. A. Khaparde, Y. Pradeep, „CIM Oriented Database for Topology Processing and Integration of Power System Applications“, Power and Energy Society General Meeting (PES), 2013, ISSN: 1932-5517, DOI: 10.1109/PESMG.2013.6672164.

[23] H. Darwen, „The Relational Model: Beginning of an Era“, Annals of the History of Computing, IEEE, Vol. 34, No. 4, 2012, ISSN: 1058-6180, DOI: 10.1109/MAHC.2012.50.

[24] I. Antović, S. Vlajić, M. Milić, D. Savić, V. Stanojević, „Model and software tool for automatic generation of user interface based on use case and data model“, Software, IET, Vol. 6, No. 6, 2012, ISSN: 1751-8806, DOI: 10.1049/iet-sen.2011.0060.

[25] I. I. Lendak, A. M. Erdeljan, D. S. Popović, „Algorithm for cataloging topologies in the Common Information Model (CIM)“, Computers and Mathematics with Applications, Vol. 61, No. 3, 2011, DOI: 10.1016/j.camwa.2010.12.021.

[26] I. Luković, „From the Synthesis Algorithm to the Model Driven Transformations in Database Design“, Proceedings of 10th International Scientific Conference on

Informatics (Informatics 2009), Herlany, Slovakia, ISBN 978-80-8086-126-1, pp. 9-18, 2009.

[27] I. Luković, A. Popović, J. Mostić, S. Ristić, „A Tool for Modeling Form Type Check Constraints and Complex Functionalities of Business Applications“, Computer Science and Information Systems (ComSIS), Consortium of Faculties of Serbia and Montenegro, Belgrade, Serbia, Vol. 7, No. 2, pp. 359-385, 2010, ISSN: 1820-0214, DOI:10.2298/CSIS1002359L.

[28] I. Luković, M. J. V. Pereira, N. Oliveira, D. Cruz, P. R. Henriques, „A DSL for PIM Specifications: Design and Attribute Grammar based Implementation“, Computer Science and Information Systems (ComSIS), Consortium of Faculties of Serbia and Montenegro, Belgrade, Serbia, Vol. 8, No. 2, pp. 379-403, 2011, ISSN: 1820-0214, DOI: 10.2298/CSIS101229018L.

[29] IEC 61970 Energy management system application program interface (EMS-API) – Part 301: Common Information Model (CIM) Base. IEC, Edition 2.0, 2007.

[30] J. Bezivin, O. Gerbe, „Towards a precise definition of the OMG/MDA framework“, Proceedings of the 16th Annual International Conference on Automated Software Engineering (ASE), 2001, ISBN: 0-7695-1426-X, ISSN: 1938-4300, DOI: 10.1109/ASE.2001.989813.

[31] J. P. A. Almeida, M. E. Iacob, P. Eck, „Requirements traceability in model-driven development: Applying model and transformation conformance“, Information Systems Frontiers, Vol. 9, No. 4, pp 327–342, 2007, DOI: 10.1007/s10796-007-9038-3.

[32] J. P. Britton, A. N. deVos, „CIM-Based Standards and CIM Evolution“, IEEE Transactions on Power Systems, Vol. 20, No. 2, 2005, ISSN: 0885-8950, DOI: 10.1109/TPWRS.2005.846202.

[33] J. P. Britton, P. Brown, J. Moseley, M. Bunda, „Optimizing Operations with CIM: Today's Grid Relies on Network Analysis (and a Lot of Data) “, Power and Energy Magazine, IEEE, Vol 14, No. 1, 2016, ISSN: 1540-7977, DOI: 10.1109/MPE.2015.2481783.

[34] J. W. Tukey, „We need both exploratory and confirmatory“, The American Statistician, Vol. 34, No. 1, pp. 23-25, 1980, DOI: 10.2307/2682991.

[35] K. Fertalj, D. Kalpic, V. Mornar, „Source code generator based on a proprietary specification language“, Proceedings of the 35th Annual Hawaii International Conference on System Sciences (HICSS), 2002, ISBN: 0-7695-1435-9, DOI: 10.1109/HICSS.2002.994498.

[36] K. Ma, B. Yang, A. Abraham, „A Template-based Model Transformation Approach for Deriving Multi-Tenant SaaS Applications“, Acta Polytechnica Hungarica, Vol. 9, No. 2, 2012.

[37] L. Ablonskis, L. Nemuraitė, „Discovery of Model Implementation Patterns in Source Code“, Information Technology and Control, Vol.39, No. 1, 2010, ISSN: 1392-124X.

[38] L. S. Levy, „A metaprogramming method and its economic justification“, IEEE Transactions on Software Engineering, Vol. SE-12, No. 2, 1986, ISSN: 0098-5589, DOI: 10.1109/TSE.1986.6312943.

[39] M. Čelković, I. Luković, S. Aleksić, V. Ivančević, „A MOF based Meta-Model and a Concrete DSL Syntax of IIS*Case PIM Concepts“, Computer Science and Information

Systems (ComSIS), Consortium of Faculties of Serbia and Montenegro, Belgrade, Serbia, Vol. 9, No. 3, pp. 1075-1103, 2012, ISSN: 1820-0214, DOI: 10.2298/CSIS120203034C.

[40] M. J. Varanda Pereira, M. Mernik, D. Cruz, P. R. Henriques, „VisualLISA: a Visual Interface for an Attribute Grammar based Compiler-Compiler“, Proceedings of 2nd Conference on Compilers, Related Technologies and Applications (CoRTA08), IPB, Bragança, Portugal, pp. 265-289, 2008.

[41] M. Mernik, J. Heering, A. M. Sloane, „When and how to develop domain-specific languages“, ACM Computing Surveys (CSUR), Vol. 37, No. 4, pp. 316-344, 2005, DOI: 10.1145/1118890.1118892.

[42] M. S. Kumar, J. Prabhu, „Comparison of NoSQL Database and Traditional Database-An emphatic analysis“, JOIV: International Journal on Informatics Visualization, ISSN: 2549-9610, Vol. 2, No. 2, 2018, DOI: 10.30630/joiv.2.2.58.

[43] Microsoft Corporation, .NET Framework, https://microsoft.com/net

[44] N. B. Hargreaves, „Novel processes for smart grid information exchange and knowledge representation using the IEC common information model“, Department of Electronic and Computer Engineering, Brunel University, 2013.

[45] N. B. Hargreaves, S. M. Pantea, G. A. Taylor, „Adopting the IEC Common Information Model to Enable Smart Grid Interoperability and Knowledge Representation Processes“, Green Energy and Technology, 2014, ISBN: 978-981-4585-29-3, DOI: 10.1007/978-981-4585-30-9_17.

[46] N. Leavitt, „Will NoSQL Databases Live Up to Their Promise? “, Computer, Vol. 43, No. 2, 2010, ISSN: 0018-9162, DOI: 10.1109/MC.2010.58.

[47] N. Nedić, G. Švenda, „Workflow Management System for DMS“, Information Technology and Control, Vol. 42, No. 4, 2013, ISSN: 1392-124X, DOI: 10.5755/j01.itc.42.4.4546.

[48] N. Obrenović, A. Popović, S. Aleksić, I. Luković, „Transformations of Check Constraint PIM Specifications“, Computer and Informatics (CAI), Slovak Academy of Sciences, Institute of Informatics, ISSN: 1335-9150, Bratislava, Slovakia, Vol. 31, No. 5, pp. 1045-1079, 2012.

[49] N. Oliveira, M. J. Varanda Pereira, P. R. Henriques, D. Cruz, B. Cramer, „VisualLISA: A Visual Environment to Develop Attribute Grammars“, Computer Science an Information Systems Journal, Special Issue on Compilers, Related Technologies and Applications (ComSIS), I. Lukovic, and A. Leitão, B. Slivnik (Guest Eds.), ISSN:1820-0214, Vol. 7, No. 2, pp. 265-289, 2010, DOI: 10.2298/CSIS1002265O.

[50] NHibernate Community, NHibernate, http://nhibernate.info/

[51] OMG Unified Modeling Language (OMG UML) Superstructure, V2.1.2. OMG Available Specification formal/2007-11-02, 2007, http://www.omg.org.

[52] Oracle Corporation, Oracle Database 11g Express Edition, http://www.oracle.com/technetwork/database/database-technologies/express-edition/overview/index.html

[53] S. Dević, B. Atlagić, Z. Gorečan, „Database modelling and development of code generator for handling power grid CIM models“, ICEST conference, ISBN: 978-86-6125-033-0, Vol. 03, pp. 865-868, 2011.

[54] S. Dević, I. Luković, „Development of a Database for the Common Information Model of Power Grids“, Information Technology and Control, Vol. 46, No. 3, 2017, ISSN: 1392-124X, DOI: 10.5755/j01.itc.46.3.14340.

[55] S. Kent, „Model driven engineering“, Proceedings of the Third International Conference on Integrated Formal Methods (IFM 2002), ISBN: 3-540-43703-7, Vol. 2335, pp. 286–298, DOI: 10.1007/3-540-47884-1_16.

[56] S. Schulte, R. Berbner, R. Steinmetz, M. Uslar, „Implementing and Evaluating the Common Information Model in a Relational and RDF-based Database“, Information Technologies in Environmental Engineering, Environmental Science and Engineering, ISSN 1863-5520, pp. 109-118, 2007, DOI: 10.1007/978-3-540-71335-7_13.

[57] V. C. Nguyen, X. Qafmolla, K. Richta, „Domain Specific Language Approach on Model-driven Development of Web Services“, Acta Polytechnica Hungarica, Vol. 11, No. 8, 2014.

[58] V. Ivančević, „Poređenje skupova podataka pomoću grafova“, Fakultete tehničkih nauka, Novi Sad, 2017.

[59] X. He, G. Wang, J. Zhao, H. Li, „The Modeling of Power Data Warehouse based on CIM“, International Conference on Power System Technology, ISBN: 1-4244-0110-0, 2006, DOI: 10.1109/ICPST.2006.321913.

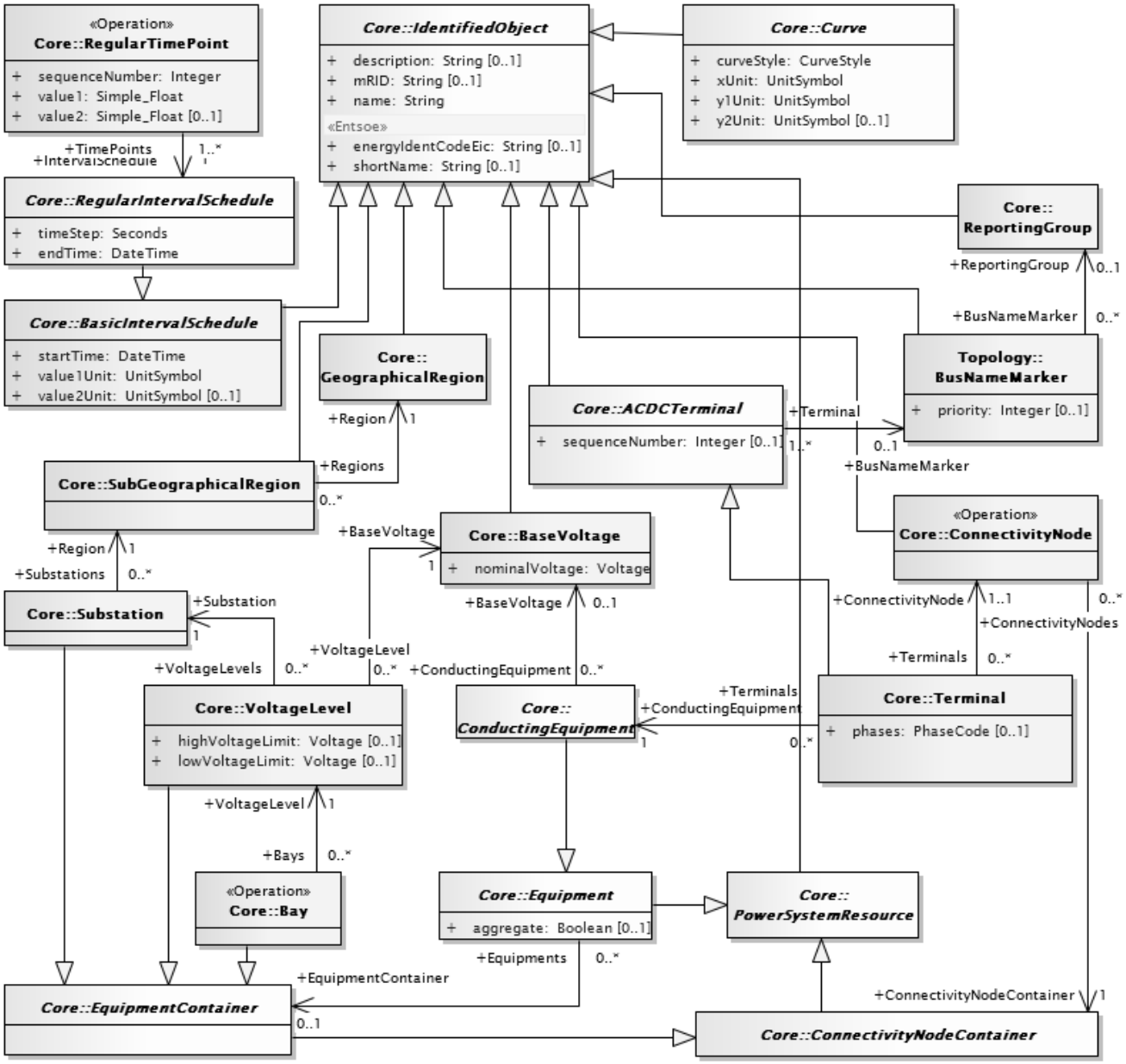
[60] Y. Pradeep, P. Seshuraju, S. A. Khaparde, R. K. Joshi, „CIM-Based Connectivity Model for Bus-Branch Topology Extraction and Exchange“, IEEE Transactions on Smart Grid, ISSN: 1949-3053, Vol 2, No. 2, 2011, DOI: 10.1109/TSG.2011.2109016.

[61] Институт за стандардизацију Србије, SRPS EN 61968-11:2014, Службени гласник: 60/14, Издање: 2 ED, 2014.

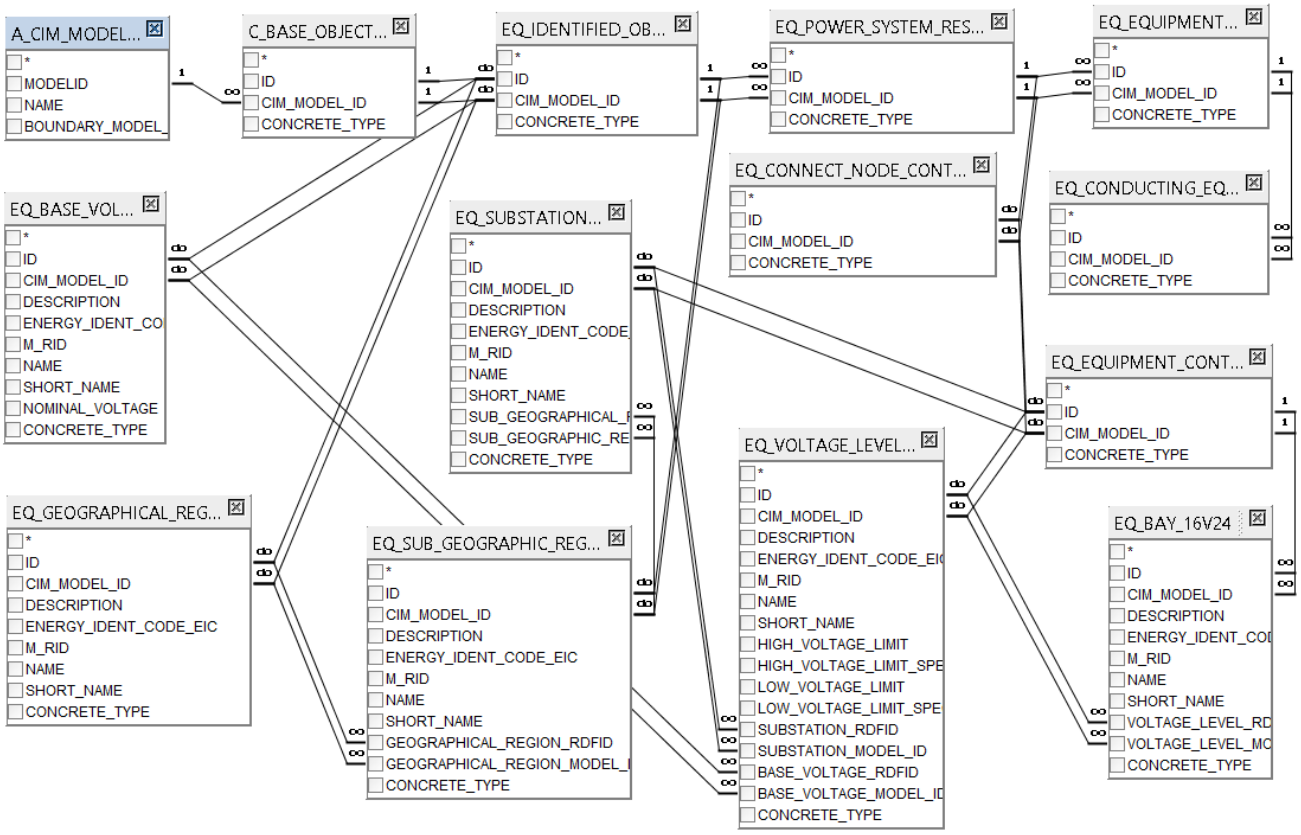
[62] Институт за стандардизацију Србије, SRPS EN 61970-301:2014, Службени гласник: 60/14, Издање: 3 ED, 2014.

Прилози

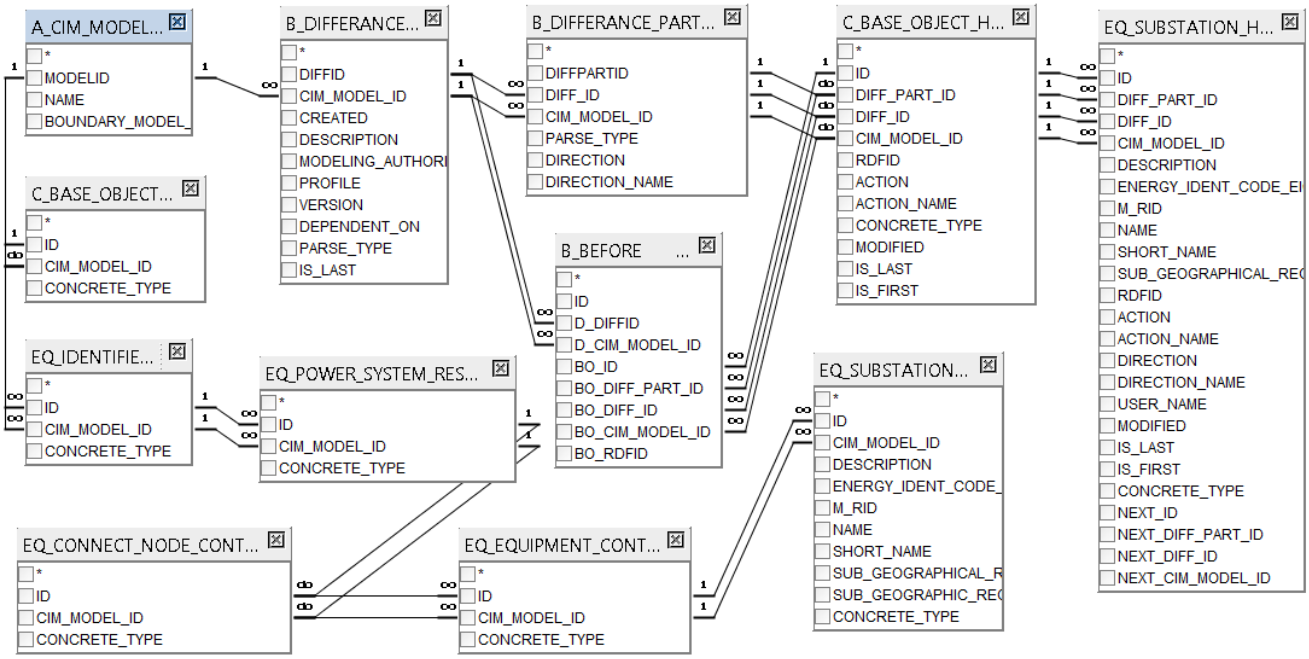
Прилог 1: Исечак CIM UML дијаграма класа EQ профила



Прилог 2: Исечак CIM шеме базе података активног модела



Прилог 3: Исечак CIM шеме базе података историјског модела



Прилог 4: Исечак пресликавања класе на шему базе података, активни модел

```
<?xml version="1.0" encoding="utf-8"?>
<hibernate-mapping namespace="CIMCommon" assembly="CIMModelCommon">
  <class name="BaseObject" table="C_Base_Object">
    <composite-id>
      <key-property name="ID" />
      <key-many-to-one name="Model" class="CIMModel_v16.CimModel16, CIMModel_v16"
        column="Cim_Model_ID" />
    </composite-id>
    <property name="ConcreteType" column="Concrete_Type" />
  </class>
</hibernate-mapping>
```

Прилог 5: Исечак пресликавања класе на шему базе података, историјски модел

```
<?xml version="1.0" encoding="utf-8"?>
<hibernate-mapping namespace="CIMModel_v16.Extension"
assembly="CIMModel_v16.Extension">
  <class name="BaseObjectDiff" table="C_BASE_OBJECT_DIFF">
    <composite-id>
      <key-property name="BaseObjectDiffID" column="ID" />
      <key-many-to-one name="DifferancePart"
class="CIMModel_v16.Extension.DifferancePart">
        <column name="Diff_Part_ID" />
        <column name="Diff_ID" />
        <column name="Cim_Model_ID" />
      </key-many-to-one>
    </composite-id>
    <property name="RDFID" column="RDFID" />
    <property name="Action" column="Action" />
    <property name="ActionName" column="Action_Name" />
    <property name="ConcreteType" column="Concrete_Type" />
    <property name="Modified" column="Modified" />
    <property name="IsLast" column="Is_Last" />
    <property name="IsFirst" column="Is_First" />
  </class>
</hibernate-mapping>
```

Прилог 6: Исечак алгоритма за поништавање појединачне измене

```
CimModel cimModel = ReadModel(modelId);

List<BaseObjectDiff> slice = GetSlice(cimModel, diffId);

List<Object> adds = new List<Object>();
List<Object> edits = new List<Object>();
List<Object> removes = new List<Object>();

foreach (BaseObjectDiff boDiff in slice) {

   BaseObject boInActive = null;
   cimModel.ElementsDictionary.TryGetValue(boDiff.RDFID, out boInActive);

   switch (boDiff.Action) {
      case ActionEnum.New:
         if (boDiff.IsFirst) {
            if (boInActive != null) removes.Add(boInActive);
         }
         else {
            if (boInActive != null) edits.Add(boDiff.CreateOriginal());
            else adds.Add(boDiff.CreateOriginal());
         }
      break;
      case ActionEnum.Update:
         if (boInActive != null) edits.Add(boDiff.CreateOriginal());
         else adds.Add(boDiff.CreateOriginal());
      break;
      case ActionEnum.Delete:
         if (boDiff.IsFirst) {
            if (boInActive != null) {
               removes.Add(boInActive);
               adds.Add(boDiff.CreateOriginal());
            }
            else adds.Add(boDiff.CreateOriginal());
         }
         else {
            if (boInActive != null) removes.Add(boInActive);
         }
      break;
   }
}

UpdateModel (cimModel, adds, removes, edits);
```