



УНИВЕРЗИТЕТ У НОВОМ САДУ  
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У  
НОВОМ САДУ

---



Душан Суђић

# **Алат за визуелно мапирање података електроенергетске мреже у CIM модел**

ДИПЛОМСКИ РАД  
- Основне академске студије -

Нови Сад, 2024.




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

## КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, <b>РБР</b> :		
Идентификациони број, <b>ИБР</b> :		
Тип документације, <b>ТД</b> :	Монографска документација	
Тип записа, <b>ТЗ</b> :	Текстуални штампани рад	
Врста рада, <b>ВР</b> :	Завршни (Bachelor) рад	
Аутор, <b>АУ</b> :	Душан Суђић	
Ментор, <b>МН</b> :	др Жељко Вуковић	
Наслов рада, <b>НР</b> :	Алат за визуелно мапирање података електроенергетске мреже у СИМ модел	
Језик публикације, <b>ЈП</b> :	Српски	
Језик извода, <b>ЈИ</b> :	Српски	
Земља публиковања, <b>ЗП</b> :	Република Србија	
Уже географско подручје, <b>УГП</b> :	Војводина	
Година, <b>ГО</b> :	2024	
Издавач, <b>ИЗ</b> :	Ауторски репринт	
Место и адреса, <b>МА</b> :	Нови Сад, Трг Доситеја Обрадовића 6	
Физички опис рада, <b>ФО</b> : (поглавља/страна/ цитата/табела/слика/графика/прилога)	8/37/23/0/17/0/0	
Научна област, <b>НО</b> :		
Научна дисциплина, <b>НД</b> :		
Предметна одредница/Кључне речи, <b>ПО</b> :	XML, мапирање података, СИМ модел, електроенергетска мрежа	
<b>УДК</b>		
Чува се, <b>ЧУ</b> :	Библиотека Факултета техничких наука, Трг Д. Обрадовића 6, Нови Сад	
Важна напомена, <b>ВН</b> :		
Извод, <b>ИЗ</b> :	У овом раду је представљен развој десктоп апликације намењене за уређивање XML докумената путем којих се врши мапирање података електроенергетске мреже у СИМ модел.	
Датум прихватања теме, <b>ДП</b> :		
Датум одбране, <b>ДО</b> :		
Чланови комисије, <b>КО</b> :		
Председник:	др Стеван Гостојић, редовни професор	Потпис ментора
Члан:	др Марко Марковић, доцент	
Члан, ментор:	др Жељко Вуковић, доцент	

Accession number, <b>ANO</b> :		
Identification number, <b>INO</b> :		
Document type, <b>DT</b> :	Monographic publication	
Type of record, <b>TR</b> :	Textual printed material	
Contents code, <b>CC</b> :	Bachelor Thesis	
Author, <b>AU</b> :	Dušan Sudić	
Mentor, <b>MN</b> :	Željko Vuković	
Title, <b>TI</b> :	A tool for visual data mapping of the power grid in the CIM model	
Language of text, <b>LT</b> :	Serbian	
Language of abstract, <b>LA</b> :	Serbian	
Country of publication, <b>CP</b> :	Republic of Serbia	
Locality of publication, <b>LP</b> :	Vojvodina	
Publication year, <b>PY</b> :	2024	
Publisher, <b>PB</b> :	Author's reprint	
Publication place, <b>PP</b> :	Faculty of Technical Sciences, Trg Dositeja Obradovića 6, Novi Sad	
Physical description, <b>PD</b> : (chapters/pages/ref./tables/pictures/graphs/appendixes)	8/37/23/0/17/0/0	
Scientific field, <b>SF</b> :		
Scientific discipline, <b>SD</b> :		
Subject/Key words, <b>S/KW</b> :	XML, data mapping, CIM model, power grid	
<b>UC</b>		
Holding data, <b>HD</b> :	The Library of Faculty of Technical Sciences, Novi Sad, Serbia	
Note, <b>N</b> :		
Abstract, <b>AB</b> :	This paper presents the development of a desktop application designed for editing XML documents used for mapping power grid data into the CIM model.	
Accepted by the Scientific Board on, <b>ASB</b> :		
Defended on, <b>DE</b> :		
Defended Board, <b>DB</b> :	President:	dr Stevan Gostojić, professor
	Member:	dr Marko Marković, assist. prof.
	Member, Mentor:	dr Željko Vuković, assist. prof.
		Menthor's sign

	УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Број:
	<b>ЗАДАТАК ЗА ИЗРАДУ ДИПЛОМСКОГ РАДА</b>	Датум:

(Податке уноси предметни наставник - ментор)

Врста студија:	а) Основне академске студије б) Основне струковне студије
Студијски програм:	Рачунарство и аутоматика
Руководилац студијског програма:	проф. др Милан Рапаић

Студент:	Душан Суђић	Број индекса:	RA 81/2020
Област:			
Ментор:	др Жељко Вуковић		

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

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

### НАСЛОВ ЗАВРШНОГ (BACHELOR) РАДА:

Алат за визуелно мапирање података електроенергетске мреже у CIM модел

### ТЕКСТ ЗАДАТКА:

1. Анализирати стање у области.
2. Изградити спецификацију захтева софтверског решења.
3. Изградити спецификацију дизајна софтверског решења.
4. Имплементирати софтверско решење према израђеној спецификацији.
5. Тестирати имплементирано софтверско решење.
6. Документовати (1), (2), (3), (4) и (5).

Руководилац студијског програма	Ментор рада

Примерак за: о - Студента; о - Ментора

## **ЗАХВАЛНОСТ**

Захвалност дугујем ментору др Жељку Вуковићу, као и стручним сарадницима Марку Лукићу, Давору Панићу и Дарку Ђуричићу на указаној стручној помоћи и саветима током израде овог рада.

Такође, захваљујем се компанији Schneider Electric на пруженој прилици за реализацију овог пројекта.

## САДРЖАЈ

<b>1.</b>	<b>УВОД</b>	<b>1</b>
<b>2.</b>	<b>ПРЕГЛЕД СЛИЧНИХ АПЛИКАЦИЈА И КОРИШЋЕНИХ СОФТВЕРСКИХ ТЕХНОЛОГИЈА</b>	<b>2</b>
2.1	Преглед сличних апликација	2
2.2	Преглед коришћених софтверских технологија	3
<b>3.</b>	<b>СПЕЦИФИКАЦИЈА ЗАХТЕВА</b>	<b>4</b>
3.1	Функционални захтеви	4
<b>4.</b>	<b>СПЕЦИФИКАЦИЈА ДИЗАЈНА</b>	<b>6</b>
4.1	Модел података	6
4.2	Архитектура система	8
4.2.1	Улази	8
4.2.2	MVVM (Model – View - ViewModel)	9
4.2.3	Излаз	9
<b>5.</b>	<b>ИМПЛЕМЕНТАЦИЈА СИСТЕМА</b>	<b>10</b>
5.1	Учитавање модела	10
5.2	Отварање XML документа	12
5.3	Додавање новог елемента у документ	14
5.4	Брисање и замена елемената	16
5.5	Дефинисање функција	17
5.6	Генерисање излазног документа	18
<b>6.</b>	<b>ДЕМОНСТРАЦИЈА</b>	<b>20</b>
6.1	Приказ документа	20
6.2	Уређивање документа	22
<b>7.</b>	<b>ЗАКЉУЧАК</b>	<b>28</b>
<b>8.</b>	<b>ЛИТЕРАТУРА</b>	<b>29</b>

## 1. УВОД

Средином двадесетог века у свету технологије и софтвера постојао је проблем неусаглашености техничких документација као и начина означавања података између различитих софтверских компанија и производа. Овај проблем је узроковао општу потребу за стварањем стандардизованих формата за размену и означавање података и први корак на путу ка решењу је начинила компанија IBM [1]. Тако настаје GML [2] (енгл. *Generalized Markup Language*), а мало касније и SGML [3] (енгл. *Standard Generalized Markup Language*) који су тада служили као универзални језици за означавање података, а данас имају само историјски значај као претече XML [4] (енгл. *Extensible Markup Language*) језика, усвојеног као стандардни формат за означавање података и њихово структурирање. XML 1.0 настаје 10. фебруара 1998. године са циљем да буде погодан формат за размену података путем мреже, формалан и прецизан, али истовремено и једноставан и лако читљив и за човека и за рачунар. Данас XML има веома широку примену и заступљеност у софтверској индустрији, и због тога се јавља потреба за бројним алатима и софтверима који човеку помажу у раду са XML документима и кодом.

Иако је XML замишљен као језик лако разумљив и читљив за човека, обимни XML документи могу постати веома компликовани за разумевање и рад над њима. За писање XML кода неопходно је познавати језичку синтаксу, као и специфичности везане за XML шему [5] на основу које се формира документ. XML шема и екстерне апликације које користе XML документ могу наметнути сложена правила на основу којих се тај документ мора писати и уређивати, а која могу бити компликована за разумевање и захтевати време за учење и навикавање од стране човека.

У овом раду је описан процес развоја и функционисање једне *desktop* апликације која кориснику омогућава једноставнији и бржи рад на XML документу, намењеном за мапирање података електроенергетске мреже у CIM [6] (енгл. *Common Information Model*) model. Апликација за циљ има да својим корисницима убрза и поједностави уређивање XML докумената, без оптерећивања корисника сложеним правилима, имплементационим и техничким детаљима самог документа и његовом интеграцијом са остатком система.

Остатак рада је организован на следећи начин.

У другом поглављу је дат преглед сличних апликација и коришћених софтверских технологија у оквиру апликације представљене у раду.

У трећем поглављу је наведена спецификација захтева која подразумева опис функционалних захтева које апликација мора да обезбеди.

У четвртном поглављу је дефинисана спецификација дизајна, у оквиру које су детаљније описани модел података и архитектура система.

У петом поглављу детаљно је објашњена имплементација система и начин функционисања кључних делова апликације.

У шестом поглављу је приказан ток интеракције корисника са апликацијом у неколико сценарија коришћења

Седмо поглавље доноси закључна разматрања и правце будућих истраживања.

## 2. ПРЕГЛЕД СЛИЧНИХ АПЛИКАЦИЈА И КОРИШЋЕНИХ СОФТВЕРСКИХ ТЕХНОЛОГИЈА

У овом поглављу су наведена и објашњена сродна решења за креирање XML шема, уређивање XML докумената, као и њихову валидацију, а потом је дат преглед коришћених софтверских технологија при изради решења представљеног у наредним поглављима.

### 2.1 Преглед сличних апликација

Овај одељак доноси преглед апликација и алата који корисницима нуде могућности креирања XML докумената и њихових шема, као и складиштење података у самим документима организованих по правилима које дефинише изабрана XML шема.

**XMLSpy** [7] је софтверски алат произведен од стране компаније Altova [8] и намењен је моделовању система креирањем шема података а потом и уређивањем XML и JSON [9] структура података креираних на основу одговарајућих шема. XMLSpy својим корисницима нуди графички приказ шеме података уз опције проширења и измене саме шеме. Такође, омогућава уређивање XML докумената у текстуалном или графичком приказу. XMLSpy је један од најпродаванијих алата за уређивање XML и JSON структура, међутим, доступан је само у виду *desktop* апликације. Овај алат нуди прегршт могућности, али са ценом комплекснијег корисничког интерфејса који захтева одређено време за учење од стране корисника који нису најбоље упознати са XML технологијама.

**Liquid Studio** [10] креиран од стране компаније Liquid Technologies [11] је *desktop* апликација која нуди бројне алате за мапирање и трансформацију података. XML Editor [12] је само један од алата у оквиру апликације Liquid Studio и намењен је уређивању XML докумената и шема, а кориснику пружа графички преглед структура података у виду стабла, табела или резултујућег XML кода. Овај алат такође нуди и валидацију докумената као и подршку за XPath [13] упите. Иако је Liquid Studio осмишљен као алат који је веома интуитиван за коришћење, одређени број напреднијих функционалности захтева привикавање на њихову употребу.

**Oxygen XML Editor** [14] је један од најпознатијих алата за XML развој који нуди широк спектар функционалности. Произведен је од стране компаније SyncRO Soft [15] чији најпознатији производ је управо овај алат. Корисницима се нуди висок степен валидације као и помоћ вештачке интелигенције током уређивања XML докумената. Oxygen XML Editor је доступан као *desktop* апликација на различитим платформама и оперативним системима. Такође, ова апликација омогућава интеграцију са другим алатима и системима као што су базе података, контроле верзија, итд. Самим тим што се кориснику нуди широк спектар могућности у оквиру ове апликације, алат може бити сувише компликован у ситуацијама када корисник тражи једноставно решење.

**XMLBlueprint** [16] компаније SyncRO Soft је софтвер за уређивање XML докумената и њихову валидацију на основу изабране XML шеме. Апликација нуди визуелни приказ XML документа који олакшава његово креирање и уређивање. Корисницима омогућава да интуитивно манипулишу XML елементима, атрибутима и садржајем. Овај алат такође пружа аутоматско попуњавање и сугестије које додатно помажу у раду са XML кодом. За разлику од осталих познатијих алата, XMLBlueprint нема толико велику подршку и ресурсе заједнице.



## **2.2 Преглед коришћених софтверских технологија**

У оквиру развоја програмског решења коришћен је C# [17] програмски језик и Microsoft .NET Framework [18] радни оквир. За манипулацију XML документима коришћене су класе за парсирање и модификацију података из System.Xml [19] простора имена, уграђеног у .NET радни оквир. Апликација је развијана у Visual Studio [20] развојном окружењу, а као главни оквир за кориснички интерфејс коришћен је WPF [21] (енгл. Windows Presentation Foundation).

### 3. СПЕЦИФИКАЦИЈА ЗАХТЕВА

Ово поглавље садржи опис свих функционалних захтева које систем треба да подржи.

#### 3.1 Функционални захтеви

Функционални захтеви су набројани на дијаграму случајева коришћења, илустрованом на слици 1, а потом је и сваки од њих детаљније објашњен у даљем тексту.



Слика 1 - Дијаграм случајева коришћења

**Креирање новог XML документа** – случај коришћења који кориснику омогућава да у оквиру апликације од самог почетка формира структуру XML документа у складу са XML шемом која се учитава из конфигурационе датотеке. Ова опција је кориснику доступна у сваком моменту у току рада апликације, и резултује приказом почетне верзије документа, спремног за додавање структурних елемената.

**Отварање постојећег XML документа** – случај коришћења при ком корисник може да рад на неком XML документу настави од почетне, већ постојеће верзије документа. Овај случај коришћења корисник може да употреби у било ком моменту током рада апликације. Први корак је одабир почетне XML датотеке из фајл система, након чега се проверава компатибилност документа са XML шемом и структура изабраног документа се приказује кориснику у главном прозору апликације.

**Додавање новог елемента у XML документ** – случај коришћења у ком корисник може да додаје нове елементе у хијерархијску структуру XML документа и тиме тај документ мења и надограђује. Нови елемент у структури је могуће додати као део садржаја већ постојећих елемената у документу. Сваки елемент у хијерархији поседује правила по

којима се формира његов садржај, и сходно томе, кориснику се у елементима чији је садржај проширив, нуде одговарајуће опције за додавање нових елемената.

**Замена постојећег елемента у XML документу** – у овом случају коришћења кориснику је дозвољено да мења структуру XML документа тиме што ће изабрани елемент из структуре заменити неким другим, новим елементом. Корисник бира елемент у хијерархији који жели да замени новим елементом, а потом му систем нуди избор новог елемента који према правилима из XML шеме може да одмени изабрани елемент.

**Брисање постојећег елемента у XML документу** – корисник има право да брише постојеће елементе у документу, докле год не нарушава правила која су дефинисана у XML шеми.

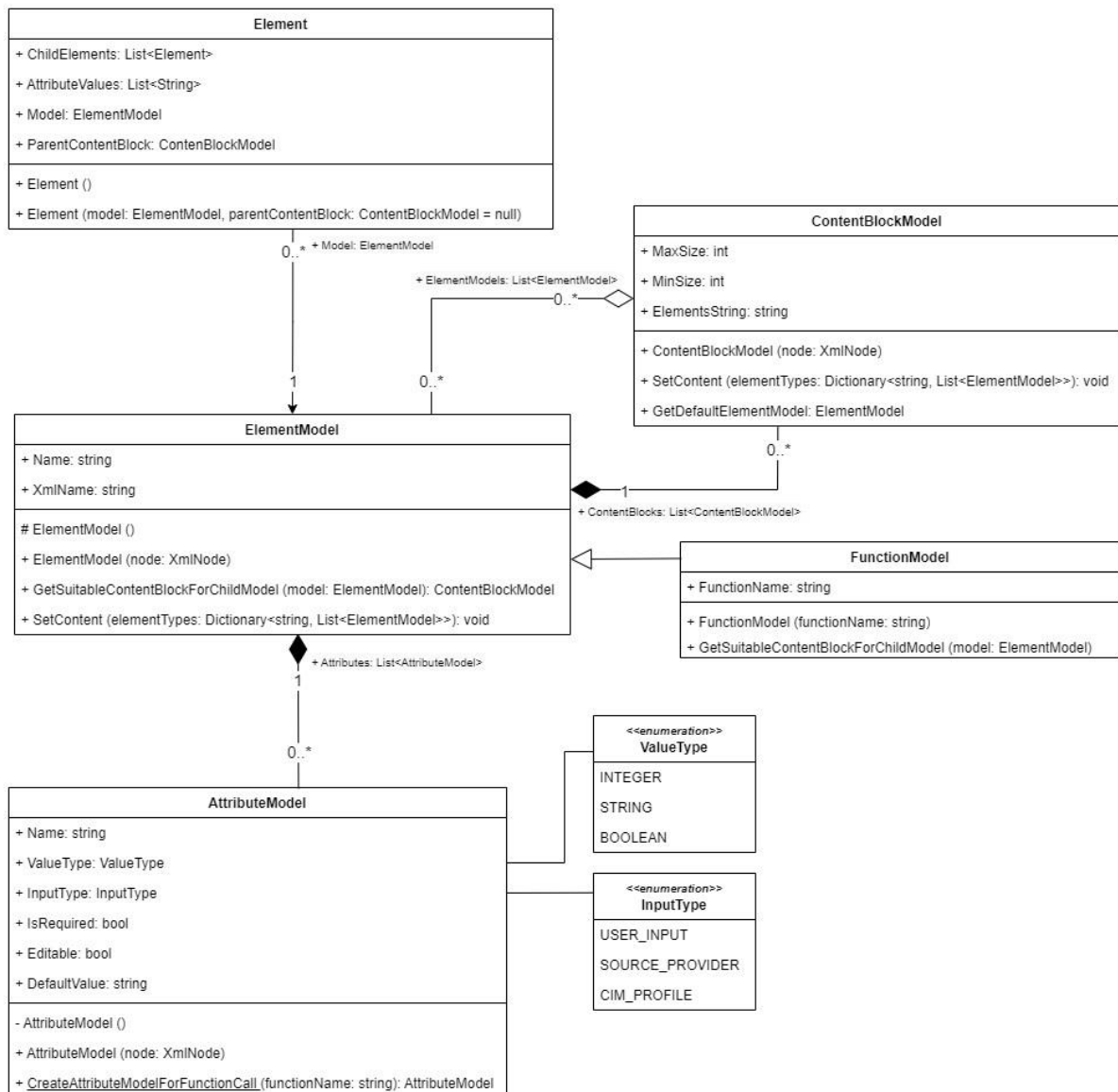
**Измена вредности атрибута елемента у XML документу** – структура сваког елемента у документу је дефинисана у XML шеми, а самим тим и списак атрибута које поседује сваки елемент. Корисник на располагању има опцију да мења вредности атрибута сваког елемента у хијерархији. Сви подаци у XML документу треба да се налазе у атрибутима елемената, док је садржај елемената намењен искључиво за дефинисање других елемената. Вредности атрибута је могуће мењати ручно, али поједини атрибути подржавају и избор вредности из екстерних датотека изабраних од стране корисника.

## 4. СПЕЦИФИКАЦИЈА ДИЗАЈНА

У овом поглављу су приказани модел података и архитектура система.

### 4.1 Модел података

Модел података на којем се заснива рад апликације је приказан на дијаграму класа, илустрованом на слици 2.



Слика 2 - Дијаграм класа

У оквиру модела података може се уочити неколико класа, кључних за функционисање система и дефинисање пословних правила. У наставку следи детаљнија анализа улоге и одговорности свих класа приказаних на дијаграму, као и веза између њих.

**ElementModel** – Класа која има централну улогу у моделу података, задужена за описивање структуре једног елемента у XML документу. Ова класа садржи податке о називу елемента, називу елемента у његовој XML репрезентацији и листу свих атрибута које елемент садржи. Опис структуре садржаја елемента одређен је редоследом блокова садржаја унутар листе ContentBlocks. ElementModel служи као шаблон за креирање конкретних елемената у документу, те се из тог разлога прослеђује као параметар конструктору класе Element. Моделовањем граматике појединачних елемената, ова класа посредно моделује и комплетну граматику излазног XML документа.

**ContentBlockModel** – Класа која описује један део (блок) садржаја елемента. Сваки елемент у XML документу садржи друге елементе који су организовани у блокове, чиме се дефинишу правила за формирање садржаја сваког елемента у хијерархији. Сваки блок садржаја карактерише листа ElementModel објеката, чиме се дефинише који се све елементи могу наћи у оквиру тог блока. Број појављивања елемената унутар једног блока садржаја се може ограничити одозго и одоздо употребом атрибута MaxSize и MinSize. Редослед елемената унутар једног блока садржаја није битан, међутим редослед блокова у садржају елемента јесте.

**AttributeModel** – Класа путем које се дефинишу атрибути елемената у XML документу. Сваки атрибут садржи назив, тип вредности коју очекује и начин на који се та вредност бира. Поред тога, постоје и информације о томе да ли се ради о обавезном атрибуту, да ли је вредност атрибута изменљива и о томе која је подразумевана вредност атрибута.

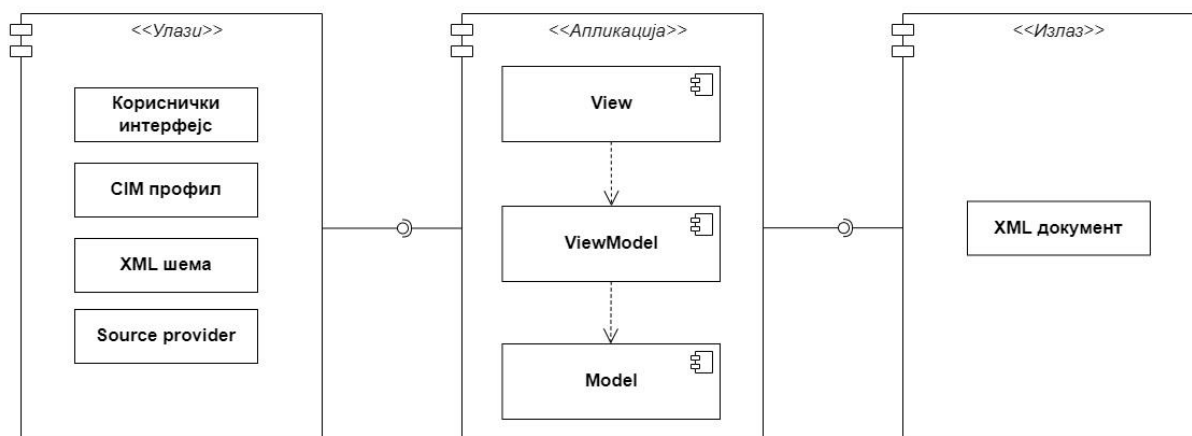
**Element** – Класа намењена за креирање конкретних инстанци елемената који ће бити преведени у њихову XML репрезентацију и записани у излазну датотеку. Инстанце Element класе се формирају на основу одговарајућег ElementModel објекта, чиме елементи се обогаћују валидационим правилима и пословном логиком. Ова класа такође садржи информације о блоку садржаја ком елемент припада у оквиру родитељског елемента, листу вредности атрибута које одговарају атрибутима из ElementModel објекта прослеђеног у конструктору и листу свих елемената потомака који формирају садржај елемента. Садржај сваке инстанце класе Element је могуће модификовати према правилима дефинисаним у листи блокова садржаја која се налази у одговарајућем ElementModel објекту. Вредности атрибута се уносе и бирају на основу правила која описује листа AttributeModel објеката, такође садржана у моделском објекту елемента. На основу једне инстанце ElementModel класе, могуће је формирати неограничен број различитих инстанци Element класе, са заједничким правилима за модификацију садржаја и вредности атрибута.

**FunctionModel** – Класа намењена као шаблон за позиве функција. Функције су један од специјалних елемената у циљном XML документу, и дефинисане су у засебном XML тагу. Намена функција се огледа у потреби да се неки елементи који се често појављују обједине у функцију, и тиме се простим позивима функција обезбеђује једноставна поновна употреба одређених делова документа. Сваким новим дефинисањем функције, потребно је обезбедити модел на основу којег ће се формирати елемент који представља позив те функције, те се у ту сврху користи управо FunctionModel класа. Ова класа наслеђује класу ElementModel и уводи додатан атрибут који носи информацију о називу функције која се позива. Због особине полиморфизма која је обезбеђена наслеђивањем класе ElementModel, поступак креирања елемента намењеног за позив

функције не захтева другачију процедуру у поређењу са креирањем свих осталих елемената.

## 4.2 Архитектура система

Архитектура система је приказана на дијаграму компоненти, илустрованом на слици 3, након чега следи детаљнији опис свих компоненти које су приказане на дијаграму.



Слика 3 - Дијаграм компоненти

### 4.2.1 Улази

**Кориснички интерфејс** – Омогућава кориснику да помоћу корисничких контрола на графичком интерфејсу апликације задаје команде, и тиме уређује излазни XML документ.

**СИМ профил** – Улазна датотека са .dll екстензијом која садржи СИМ модел описан скупом класа и атрибута чији се називи могу користити за мапирање поменутог СИМ модела на елементе у излазном XML документу. Атрибути елемената у документу који подржавају ово мапирање, нуде кориснику избор вредности из скупа вредности прочитаних из ове улазне датотеке.

**XML шема** – XML документ у којем је дата спецификација свих подржаних врста елемената који су део граматике излазног документа. Учитавањем ове датотеке на почетку рада апликације, формира се каталог елемената сачињен од објеката класе `ElementModel`, на основу којих се креирају конкретни елементи у излазном фајлу. Ова улазна датотека омогућава апликацији да избегне чврсту спрегу са граматичким правилима излазног документа што доприноси прилагодљивости апликације на измене и допуне саме граматике.

**Source provider** – XML документ са описом структуре ентитета и њихових атрибута прочитаних из екстерне базе података. Слично СИМ профили, Source provider служи за мапирање вредности атрибута елемената излазног документа на називе конкретних ентитета и атрибута садржаних у овој датотеци. Такође, и ово мапирање је подржано за само одређене атрибуте само одређених елемената, а спроводи се избором вредности атрибута из скупа вредности које ова датотека нуди.

#### 4.2.2 MVVM (Model – View - ViewModel)

Инфраструктура саме апликације изграђена је по MVVM (Model – View -ViewModel) [22] обрасцу. MVVM је трослојни архитектурални образац који за циљ има одвајање пословне логике и доменских правила (модела) од презентационог слоја (погледа) увођењем средњег слоја као посредника у њиховој комуникацији (модела погледа).

**Модел (Model)** – слој који садржи модел података, као и пословна и валидациона правила која важе на нивоу читавог система. Овај слој је потпуно независан од презентационог слоја, а комуникацију врши искључиво са средишњим (ViewModel) слојем. На примеру конкретне апликације која је представљена у овом раду, у моделском слоју се налази модел података, приказан и објашњен у претходном поглављу, као и класе које имају улогу домеских сервиса за спровођење пословне логике апликације.

**Поглед (View)** - презентациони слој који представља кориснички интерфејс и садржи ограничену логику намењену искључиво за управљање визуелним понашањем графичког интерфејса. Овај слој нема приступ подацима из моделског слоја, већ је повезан само са средишњим (ViewModel) слојем путем *data binding* [23] механизма. Поглед (View) убраја декларације изгледа свих прозора и екрана апликације као и логику за управљање њиховим понашањем и повезивањем са ViewModel слојем.

**Модел погледа (ViewModel)** – средишњи слој трослојне архитектуре који има улогу споне између моделског и презентационог слоја. Главна одговорност овог слоја је да моделске податке прилагоди приказу у презентационом слоју и да обезбеди механизме синхронизације рада преостала два слоја, како би се услед измена на једном од њих изазвала одговарајућа реакција другог и тиме одржала конзистентност система. У оквиру апликације, ViewModel слој садржи класе које презентационом слоју обезбеђују податке који се односе на визуелну презентацију појединачних елемената и њихових атрибута као и резултујућег XML документа.

#### 4.2.3 Излаз

Излаз система је XML документ генерисан на основу спецификације корисника задате путем графичког интерфејса апликације. Након што је завршио са уређивањем документа путем овог алата, корисник може да сачува измене и креирану структуру документа у излазној XML датотеци. Такође, рад апликације не мора почети од новог документа, већ је могуће наставити уређивање постојећег чиме овај документ има двојну улогу и улазне и излазне датотеке.

## 5. ИМПЛЕМЕНТАЦИЈА СИСТЕМА

У овом поглављу су представљени кључни делови имплементације софтверског система. Програмско решење представљено у овом поглављу је подељено у неколико функционалних целина: учитавање модела, отварање XML документа, додавање новог елемента у документ, брисање и замена елемената, дефинисање функција и генерисање излазног документа.

### 5.1 Учитавање модела

Како је граматика циљног XML документа конфигурисана у улазној датотеци у виду XML шеме, на почетку рада апликације неопходно је учитати ту шему и формирати каталог свих модела елемената који су подржани у граматички излазног документа. У те сврхе, користи се статичка класа `ElementModelProvider` и њена метода `LoadModel()` приказана на листингу 1.

```
public static class ElementModelProvider
{
    private static string path = "../../../Input/model.xml";
    private static List<ElementModel> ElementModels = new();
    private static Dictionary<string, FunctionModel> FunctionModels = new();
    1 reference
    public static void LoadModel()
    {
        try
        {
            Dictionary<string, List<ElementModel>> elementTypes = new();
            XmlDocument xmlDoc = new XmlDocument();
            xmlDoc.Load(path);

            XmlNodeList elementNodes = xmlDoc.SelectNodes("//Element");
            if (elementNodes != null)
                foreach (XmlNode elementNode in elementNodes)
                    ElementModels.Add(new ElementModel(elementNode));

            XmlNodeList typeNodes = xmlDoc.SelectNodes("//Type");
            if (typeNodes != null)
                foreach (XmlNode typeNode in typeNodes)
                {
                    var name = typeNode.Attributes["Name"]?.InnerText;
                    var elementModels = new List<ElementModel>();
                    string[] elementNames = typeNode.InnerText.Trim().Split(',');
                    foreach (string elementName in elementNames)
                        elementModels.Add(GetElementModelByName(elementName));
                    elementTypes.Add(name, elementModels);
                }

            foreach (var element in ElementModels)
                element.SetContent(elementTypes);
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex);
        }
    }
    7 references
    public static ElementModel GetElementModelByName(string name)
    {
        return ElementModels.FirstOrDefault(x => x.Name.Equals(name));
    }
    //...
}
```

Листинг 1 – Учитавање XML шеме



Како је XML шема сама по себи XML документ, за потребе њеног парсирања и обраде коришћене су класе XmlDocument, XmlNodeList и XmlNode из System.Xml простора имена. Први корак у поступку учитавања модела у функцији LoadModel је прикупљање свих елемената улазне шеме са тагом <Element>. Пример једног таквог елемента у шеми дат је на листингу 2.

```
<Element Name="CimClass" XMLName="CimClass">
  <Attribute Name="name" IsRequired="true" ValueType="STRING" Input="CIM_PROFILE"/>
  <Attribute Name="source" IsRequired="true" ValueType="STRING" Input="SOURCE_PROVIDER"/>
  <ContentBlock MinSize="0" MaxSize="-1">
    CimProperty,Function
  </ContentBlock>
</Element>
```

Листинг 2 – Пример елемента у XML шеми

Након прикупљања свих <Element> чворова у структури, за сваки од њих се креира објекат класе ElementModel прослеђивањем објекта класе XmlNode у конструктору, где се подаци тог објекта мапирају на поља у класи ElementModel, као што је приказано на листингу 3. На сличан начин функционишу и конструктори класа AttributeModel и ContentBlockModel.

```
public ElementModel(XmlNode node)
{
    ContentBlocks = new();
    Attributes = new();
    Name = node.Attributes["Name"]?.InnerText;
    XMLName = node.Attributes["XMLName"]?.InnerText;

    foreach (XmlNode attributeNode in node.SelectNodes("Attribute"))
        Attributes.Add(new AttributeModel(attributeNode));

    foreach (XmlNode contentBlock in node.SelectNodes("ContentBlock"))
        ContentBlocks.Add(new ContentBlockModel(contentBlock));
}
```

Листинг 3 – Конструктор класе ElementModel

Након тога, формирани објекти ElementModel класе се чувају у статичкој листи List<ElementModel> ElementModels која представља каталог елемената који се чува у класи ElementModelProvider.

Други корак у процесу учитавања модела се односи на прикупљање типова елемената из XML шеме. Као што је приказано на листингу 2, у блоковима садржаја су набројани називи ElementModel објеката које ти блокови подржавају, одвојени запетом. Понекад се може јавити потреба за обједињавањем сличних ElementModel објеката у један тип елемената. Тиме се обезбеђује једноставније дефинисање тела <ContentBlock> тага. Пример дефинисања типа елемената и његова употреба у блоку садржаја су приказани на листингу 4.

```

<Type Name="Expression">
    Constant, SetDefault, GetSourceValue, Function, GetEntityValue, BreakInstanceCreation,
    StringConcatenate, StringRemove, StringSplit, Substring, StringToLower, StringToUpper,
    StringTrim, StringTrimEnd, StringTrimStart, StringIndexOf, StringLastIndexOf, StringLength,
    StringFormat, Substraction, MathPow, Division, MathMax, MathMin, AbsoluteValue, MathSqrt,
    MathCeiling, MathFloor, MathRound, MathTruncate, MatchCos, MathSin, MathTan, Addition,
    Multiplication, IfBlock, SwitchCase, GetPrimarySource, GetSourceColumn, GetSourceTable
</Type>
<Type Name="Conditions">
    Condition, AndConditions, OrConditions
</Type>
<Element Name="If" XMLName="If">
    <ContentBlock MinSize="1" MaxSize="1">
        Conditions
    </ContentBlock>
    <ContentBlock MinSize="1" MaxSize="1">
        Expression
    </ContentBlock>
</Element>

```

Листинг 4 – Обједињавање елемената у типове

Типови елемената након читања из шеме се чувају у локалној променљивој `Dictionary<string, List<ElementModel>> elementTypes` функције `LoadModel()` као што је приказано на листингу 1. Та променљива се прослеђује сваком објекту класе `ElementModel` из каталога, како би се сваки блок садржаја попунио референцама ка објектима из каталога који одговарају том блоку. Овим се процес учитавања модела завршава и апликација је спремна за креирање документа по узору на граматику описану каталогом из класе `ElementModelProvider`.

## 5.2 Отварање XML документа

Након учитавања модела при покретању апликације, кориснику се нуди празан документ спреман за уређивање, али такође постоји и опција отварања постојећег документа из локалног фајл система и наставка рада на њему. У овом одељку је објашњено програмско решење за формирање структуре података еквивалентне XML документу којег корисник отвара на почетку рада апликације.

Читањем XML датотеке изабране од стране корисника, за сваки елемент у структури документа се тражи објекат класе `ElementModel` из каталога модела у класи `ElementModelProvider` по вредности атрибута `XMLName`. Након тога, формирају се инстанце класе `Element`, повезане са пронађеним моделом у каталогу и свака се попуњава прочитаним вредностима атрибута и рекурзивно елементима потомцима.

У оквиру улазне XML шеме, може се нагласити да одређени објекат `ElementModel` класе није предвиђен за писање у излазни документ, већ да се ради о помоћном моделу елемента. То се може постићи изостављањем вредности за атрибут `XMLName`. Конкретни елементи формиран на основу оваквих модела се „прескачу“ приликом генерисања њихове XML репрезентације, и на њихово место у структури се уписују елементи који су њихови директни потомци. Уколико се приликом отварања већ формираног документа региструје да део структуре не одговара његовој граматици, проверава се да ли постоје помоћни модели елемената у каталогу чијим додавањем би се та структура усагласила са граматичким правилима.

Кључна компонента софтверског решења за конверзију документа у еквивалентну структуру података унутар апликације је статичка класа `XmlElementFactory` и њена метода `GetElement`, приказана на листингу 5.

```
public static Element GetElement(XmlElement xmlElement, ContentBlockModel parentBlock = null)
{
    Element element = new Element();
    element.Model = ElementModelProvider.GetElementModelByXMLElement(xmlElement);
    element.ParentContentBlock = parentBlock;
    foreach (XmlAttribute attr in xmlElement.Attributes)
        element.AttributeValues.Add(attr.Value);
    addChildren(xmlElement, element);
    return element;
}

2 references
private static void addChildren(XmlElement xmlElement, Element element)
{
    foreach (XmlElement childXmlElement in xmlElement.ChildNodes)
    {
        ElementModel childModel = ElementModelProvider.GetElementModelByXMLElement(childXmlElement);
        ContentBlockModel parentContentBlock = element.Model.GetSuitableContentBlockForChildModel(childModel);
        if (parentContentBlock == null)
        {
            ElementModel supportingModel = ElementModelProvider.GetWrapperModel(childModel);
            Element supportingElement = new Element();
            supportingElement.Model = supportingModel;
            supportingElement.ParentContentBlock = element.Model.GetSuitableContentBlockForChildModel(supportingModel);
            addChildren(xmlElement, supportingElement);
            element.ChildElements.Add(supportingElement);
            break;
        }
        else
        {
            Element childElement = GetElement(childXmlElement, parentContentBlock);
            element.ChildElements.Add(childElement);
        }
    }
}
```

Листинг 5 – Креирање елемента на основу његове XML структуре

Позив методе `GetElement` при отварању документа се врши у методи `LoadFromXmlDocument` објекта класе `DocumentViewModel` који служи као модел погледа у главном прозору за приказ документа. Подразумева се да сваки документ унутар коренског елемента садржи елементе `CimClasses` и `FunctionDefinitions` који се попуњавају према граматичи из шеме. Према томе, сваки објекат `DocumentViewModel` класе садржи те елементе такође, као што је приказано на листингу 6.

```
public sealed class DocumentViewModel
{
    6 references
    public ElementViewModel CimClasses { get; set; }
    6 references
    public ElementViewModel FunctionDefinitions { get; set; }
    5 references
    public string OutputPath { get; set; }
    1 reference
    public DocumentViewModel() { }
    1 reference
    public void setup()
    {
        CimClasses = new ElementViewModel(new Element(ElementModelProvider.GetElementModelByName("CimClasses")));
        FunctionDefinitions = new ElementViewModel(new Element(ElementModelProvider.GetElementModelByName("FunctionDefinitions")));
    }
    // ...
}
```

Листинг 6 – Класа `DocumentViewModel`

На листингу 7. је дат приказ методе `LoadFromXmlDocument` из класе `DocumentViewModel`, која обједињује комплетан процес отварања документа из екстерне датотеке. Ова метода је задужена за попуњавање елемената `CimClasses` и `FunctionDefinitions` уз помоћ статичке класе `XmlElementFactory` и њене методе `GetElement`, чиме се поступак отварања постојећег документа завршава.

```
public void LoadFromXmlDocument(string path)
{
    Reset();
    OutputPath = path;
    XmlDocument document = new XmlDocument();
    document.Load(path);
    XmlNodeList functionDefinitionNodes = document.SelectNodes("//FunctionDefinitions/Function");
    if (functionDefinitionNodes != null)
    {
        foreach (XmlElement functionDefinitionNode in functionDefinitionNodes)
        {
            Element el = XmlElementFactory.GetElement(functionDefinitionNode);
            el.ParentContentBlock = FunctionDefinitions.Element.Model.ContentBlocks[0];
            ElementModelProvider.AddNewFunctionDefinition(el.AttributeValues[0]);
            FunctionDefinitions.Element.ChildElements.Add(el);
            FunctionDefinitions.ChildViewModels.Add(new ElementViewModel(el, FunctionDefinitions));
            FunctionDefinitions.SetRemovableForChildren();
        }
    }
    XmlNodeList cimClassNodes = document.SelectNodes("//CimClass");
    if (cimClassNodes != null)
    {
        foreach (XmlElement cimClassNode in cimClassNodes)
        {
            Element el = XmlElementFactory.GetElement(cimClassNode);
            el.ParentContentBlock = CimClasses.Element.Model.ContentBlocks[0];
            CimClasses.Element.ChildElements.Add(el);
            CimClasses.ChildViewModels.Add(new ElementViewModel(el, CimClasses));
            CimClasses.SetRemovableForChildren();
        }
    }
    HasUnsavedChanges = false;
}
```

Листинг 7 – Учитавање документа

### 5.3 Додавање новог елемента у документ

Након учитавања модела и отварања XML документа, корисник има опцију уређивања отвореног документа. Једна од основних операција над документом јесте његово проширење додавањем нових елемената у његову хијерархију.

Корисник нове елементе у документ може додавати само на оним местима која су дозвољена према граматичким правилима. Уколико је неки елемент у структури документа проширив другим елементима, и уколико у његовом садржају има довољно места за нови елемент, апликација кориснику нуди избор врсте елемента коју жели да дода у тај садржај. Кориснику се нуде само оне врсте елемената које се према граматици могу наћи у садржају елемента који се проширује. Прикупљање свих модела елемената који задовољавају овај критеријум се врши у методи `GetModelsForNewChildElement` класе `ElementModelProvider`, приказаној на листингу 8. У оквиру ове методе, проверава се који се модели елемената могу појавити у сваком од блокова садржаја елемента који се проширује, али и да ли у тим блоковима садржаја има довољно места за нови елемент.

```

public static List<ElementModel> GetModelsForNewChildElement(Element element)
{
    List<ElementModel> ret = new();
    foreach (var block in element.Model.ContentBlocks)
    {
        if (block.MaxSize == -1)
            ret.AddRange(block.ElementModels);
        else
        {
            int counter = 0;
            foreach (var elem in element.ChildElements)
            {
                if (block.ElementModels.Contains(elem.Model)
                    || (elem.Model is FunctionModel
                        && block.ElementModels.Contains(ElementModelProvider.GetElementModelByName("Function"))))
                {
                    counter++;
                    continue;
                }
                if (counter > 0) break;
            }
            if (counter < block.MaxSize)
                ret.AddRange(block.ElementModels);
        }
    }
    return ret;
}

```

Листинг 8 – GetModelsForNewChildElement метода

Након што корисник одабере модел за нови елемент из листе модела коју враћа метода GetModelsForNewChildElement, неопходно је креирати инстанцу класе Element на основу изабраног модела, тј објекта класе ElementModel. Конструктор класе Element којем се прослеђује изабрани модел је представљен на листингу 9.

```

public Element(ElementModel model, ContentBlockModel parentContentBlock = null)
{
    Model = model;
    ParentContentBlock = parentContentBlock;
    AttributeValues = new List<string>();
    foreach (var attr in Model.Attributes)
        AttributeValues.Add(attr.DefaultValue);

    ChildElements = new();
    foreach (var block in Model.ContentBlocks)
        for (int i = 0; i < block.MinSize; i++)
            ChildElements.Add(new Element(block.GetDefaultElementModel(), block));
}

```

Листинг 9 – Конструктор класе Element

Приликом креирања новог елемента, вредности његових атрибута се попуњавају подразумеваним вредностима за тај атрибут и садржај елемента се попуњава минималним бројем нових елемената, тако се задовољи ограничење задато атрибутом MinSize за сваки блок садржаја у моделу. Такође, елементу се приписује и информација ком родитељском блоку садржаја припада.

Након креирања елемента, он се додаје у садржај елемента којег је корисник изабрао да прошири. Ради прилагођавања података за приказ кориснику, за сваки нови елемент се креира објекат класе ElementViewModel након чега се и освежава сам приказ на којем корисник може да уочи елемент који је додао.



## 5.4 Брисање и замена елемената

Још једна битна операција над излазним документом је и брисање елемената из његове структуре. Корисник има право брисања сваког елемента у хијерархији документа, уколико се његовим брисањем не нарушава ограничење минималне величине блока садржаја којем тај елемент припада. Сваки елемент води рачуна о томе који елементи у листи његових директних потомака могу бити обрисани, а та провера се врши у објекту класе `ElementViewModel` који одговара том елементу путем методе `SetRemovableForChildren`. Преглед имплементације ове методе је дат на листингу 10.

```
public void SetRemovableForChildren()
{
    foreach (var child in ChildViewModels)
        child.IsRemovable = Element.ChildElements.
            Where(c => c.ParentContentBlock == child.Element.ParentContentBlock).
            ToList().Count > child.Element.ParentContentBlock.MinSize;
}
```

Листинг 10 – Метода `SetRemovableForChildren`

Након што корисник обрише елемент, из хијерархије се уклања и сав његов садржај тј. сви његови потомци. Сваким брисањем елемента, поново се позива метода `SetRemovableForChildren` у моделу погледа родитељског елемента, како би се изнова прорачунало који елементи у структури се могу обрисати према правилима граматике.

Уколико елемент не може да се обрише из садржаја другог елемента због ограничења минималне величине блока садржаја ком припада, кориснику се нуде опције да тај елемент замени неким другим. Слично као при додавању новог елемента у документ, кориснику ће бити понуђени само они модели елемената који се према граматичким правилима могу наћи на месту елемента који се замењује. Издајање модела елемената из каталога који одговарају овом критеријуму се врши у класи `ElementModelProvider` и њеној методи `GetReplacableModelsForElement` приказаној на листингу 11.

```
public static List<ElementModel> GetReplacableModelsForElement(Element element)
{
    if (element.ParentContentBlock == null)
        return null;
    if (element.ParentContentBlock.ElementModels.Count == 1)
        return null;
    var list = element.ParentContentBlock.ElementModels.Where(e => !e.Name.Equals(element.Model.Name)).ToList();
    return list;
}
```

Листинг 11 – Претрага модела из каталога за замену елемента

Одабиром модела елемента којим ће изабрани елемент бити замењен, врши се брисање елемента који се замењује и додавање новог елемента креираног на основу изабраног модела на његово место.

## 5.5 Дефинисање функција

Функције су елементи са `<Function>` тагом у излазном документу. Функције које се налазе унутар `<FunctionDefinitions>` тага представљају дефиницију функције и у свом садржају обухватају друге елементе. Позиви функција су елементи са `<Function>` тагом који се налазе ван `<FunctionDefinitions>` елемента, а односе се на функције које су дефинисане унутар њега. Позиви функција имају празан садржај, а у свом атрибуту `Name` носе информацију о називу функције која се позива.

Дефиниције функција имају двојну улогу у систему пошто представљају и конкретан елемент у документу али и модел за елементе који представљају њен позив. Због тога, неопходно је осим каталога свих модела елемената из шеме водити евиденцију и о доступним моделима за позиве функција које су дефинисане у `<FunctionDefinitions>` секцији документа. Софтверска компонента која репрезентује модел за позиве функција је класа `FunctionModel`, наследница класе `ElementModel`, и приказана је на листингу 12.

```
public class FunctionModel : ElementModel
{
    4 references
    public string FunctionName { get; private set; }
    2 references
    public FunctionModel(string functionName) : base()
    {
        FunctionName = functionName;
        Name = "FunctionCall";
        XMLName = "Function";
        ContentBlocks = new();
        Attributes = [AttributeModel.CreateAttributeModelForFunctionCall(functionName)];
    }
    5 references
    public override ContentBlockModel GetSuitableContentBlockForChildModel(ElementModel model)
    {
        return null;
    }
    2 references
    public override string ToString()
    {
        return Name + " [" + FunctionName + "];"
    }
}
```

Листинг 12 – Класа `FunctionModel`

Корисник током додавања новог елемента у документ може одабрати да дода елемент позива функције. Том приликом корисник бира коју претходно дефинисану функцију жели да позове, чиме се креира елемент прослеђивањем одговарајуће инстанце `FunctionModel` класе у његов конструктор. Како би се очувала конзистентност система, неопходно је ажурирати каталог модела за позиве функција сваки пут када се елемент дефиниције функције креира, обрише или преименује. Овај каталог је садржан у статичкој класи `ElementModelProvider` у променљивој `Dictionary<string, FunctionModel> FunctionDefinitions`. Како је овај каталог реализован у форми речника, путем имена функције се директно може приступити и моделу за креирање њеног позива.

Кључне методе класе `ElementModelProvider` за ажурирање и рад са каталогом функција су приказане на листингу 13.

```

2 references
public static void AddNewFunctionDefinition(string functionName)
{
    FunctionModels.Add(functionName, new FunctionModel(functionName));
}

2 references
public static bool FunctionNameAlreadyInUse(string functionName)
{
    return FunctionModels.ContainsKey(functionName);
}

1 reference
public static void RemoveFunctionDefinition(string functionName)
{
    FunctionModels.Remove(functionName);
}

1 reference
public static void ResetFunctionDefinitions()
{
    FunctionModels.Clear();
}

1 reference
public static void RenameFunction(string functionName, string newName)
{
    FunctionModels.Remove(functionName);
    FunctionModels.Add(newName, new FunctionModel(newName));
}

```

Листинг 13 – Методе за рад са каталогом функција

Креирањем новог елемента у структури који репрезентује дефиницију функције, додаје се нови модел за њен позив у каталог модела функција. Брисањем дефиниције функције се такође брише и одговарајући модел из каталога. Функције је могуће и преименовати чиме се ажурира одговарајући модел и пролази кроз читав документ и сваки позив те функције се ажурира са новом вредношћу за атрибут Name.

## 5.6 Генерисање излазног документа

По завршетку уређивања документа у оквиру апликације, корисник може да сачува измене у екстерној XML датотеци. Уколико је рад на документу почео од постојећег документа, чувањем измена нова верзија документа ће бити сачувана у датотеци из које је документ првобитно учитан. У супротном корисник бира датотеку из локалног фајл система у којој жели да сними измењени документ.

Приликом задавања команде за чување документа, позива се метода ExportToXmlDocument класе DocumentViewModel над објектом који репрезентује документ у главном прозору апликације. Као и до сада, за рад са XML документима су коришћене класе из System.XML простора имена. Приказ методе је дат на листингу 14.

```

public XmlDocument ExportToXmlDocument()
{
    XmlDocument xmlDoc = new XmlDocument();
    XmlElement rootElement = xmlDoc.CreateElement("Root");
    rootElement.AppendChild(XmlElementFactory.GetXmlElement(FunctionDefinitions.Element, xmlDoc));
    rootElement.AppendChild(XmlElementFactory.GetXmlElement(CimClasses.Element, xmlDoc));
    xmlDoc.AppendChild(rootElement);
    HasUnsavedChanges = false;
    return xmlDoc;
}

```

Листинг 14 – Конверзија објекта DocumentViewModel класе у XML документ



За потребе конверзије појединачних елемената у документу у њихову XML репрезентацију, коришћена је статичка класа XmlElementFactory и њена метода GetXmlElement представљена на листингу 15. Ова метода је рекурзивног карактера, тј. генерисање XML репрезентације једног елемента, позива исту методу примењену на све његове директне потомке.

```
4 references
public static XmlElement GetXmlElement(Element element, XmlDocument doc = null)
{
    if (doc == null)
    {
        doc = new XmlDocument();
        if (element.Model.XMLName.Length == 0)
            return null;
    }
    XmlElement node = doc.CreateElement(element.Model.XMLName);
    foreach (var attr in element.Model.Attributes)
        node.SetAttribute(attr.Name, element.AttributeValues[element.Model.Attributes.IndexOf(attr)]);
    appendChildNodes(element, doc, node);
    return node;
}

2 references
private static void appendChildNodes(Element element, XmlDocument doc, XmlNode node)
{
    foreach (var child in element.ChildElements)
    {
        if (child.Model.XMLName.Length > 0)
            node.AppendChild(GetXmlElement(child, doc));
        else
            appendChildNodes(child, doc, node);
    }
}
```

Листинг 15 – Конверзија елемента у еквивалентну XML репрезентацију

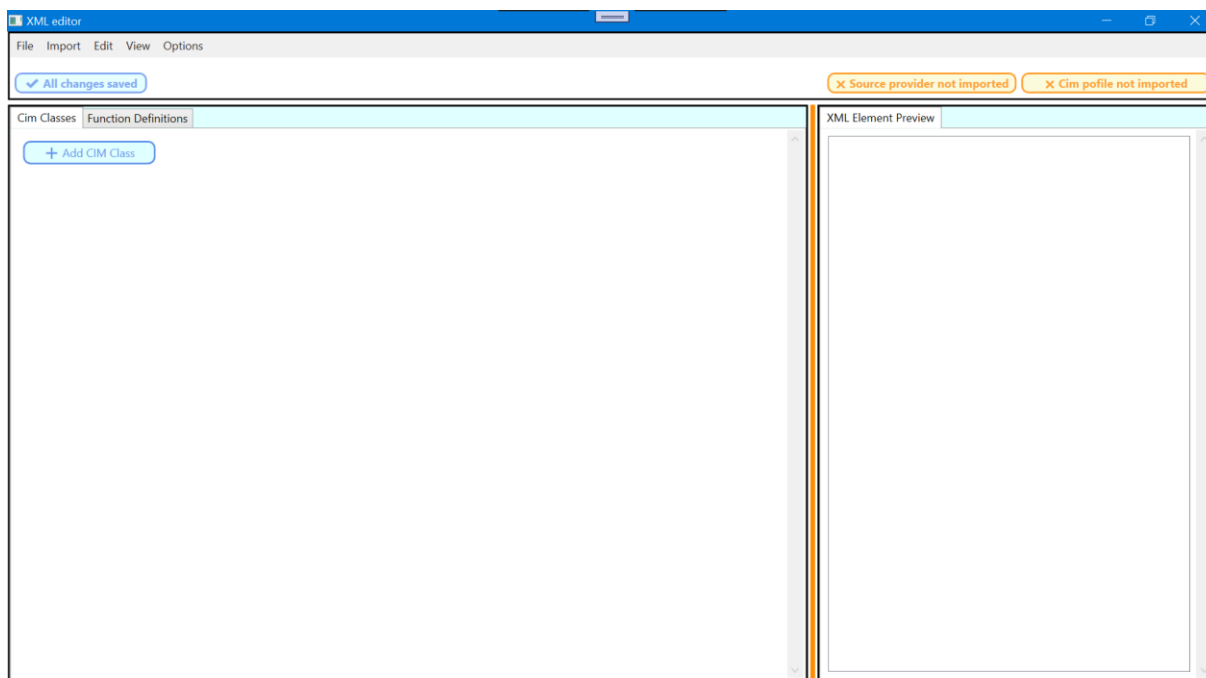
Генерисање излазног документа се може посматрати као процес инверзан процесу отварања XML документа и читавања његовог садржаја. Генерисањем излазног документа се затвара се комплетан ток рада апликације, од улаза до излаза.

## 6. ДЕМОНСТРАЦИЈА

У овом поглављу је представљен и описан ток интеракције корисника са апликацијом у неколико случајева коришћења.

### 6.1 Приказ документа

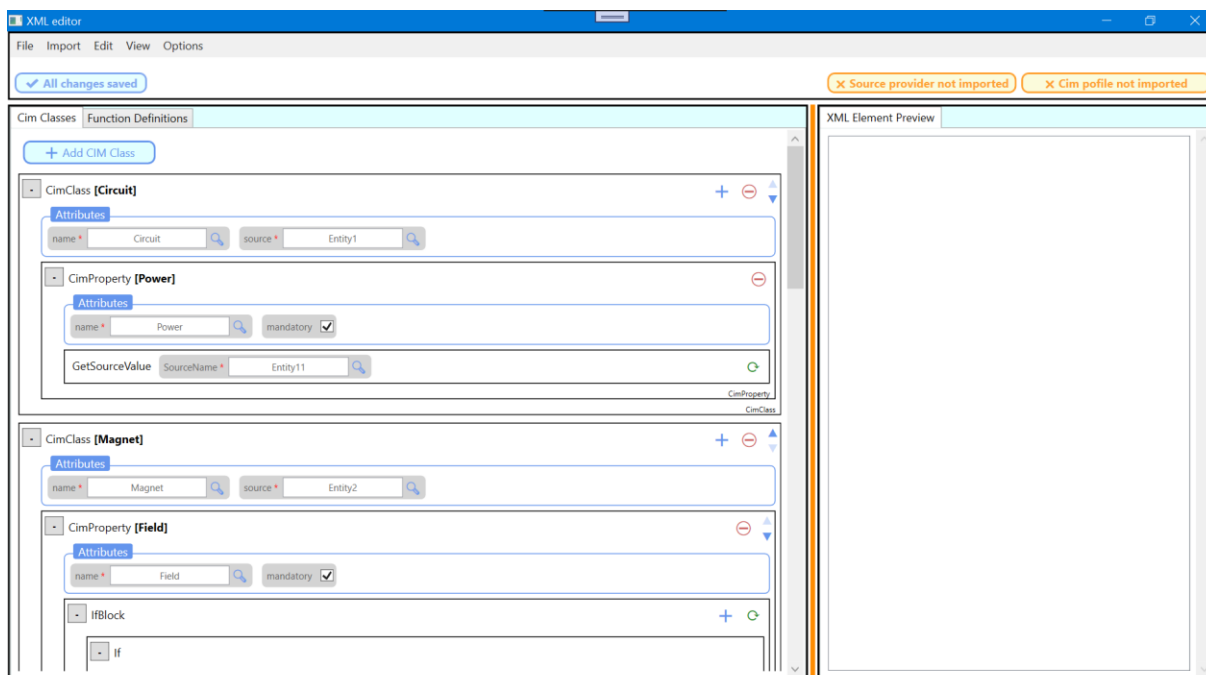
При покретању апликације, кориснику се приказује главни прозор, приказан на слици 4.



Слика 4 – Главни прозор апликације

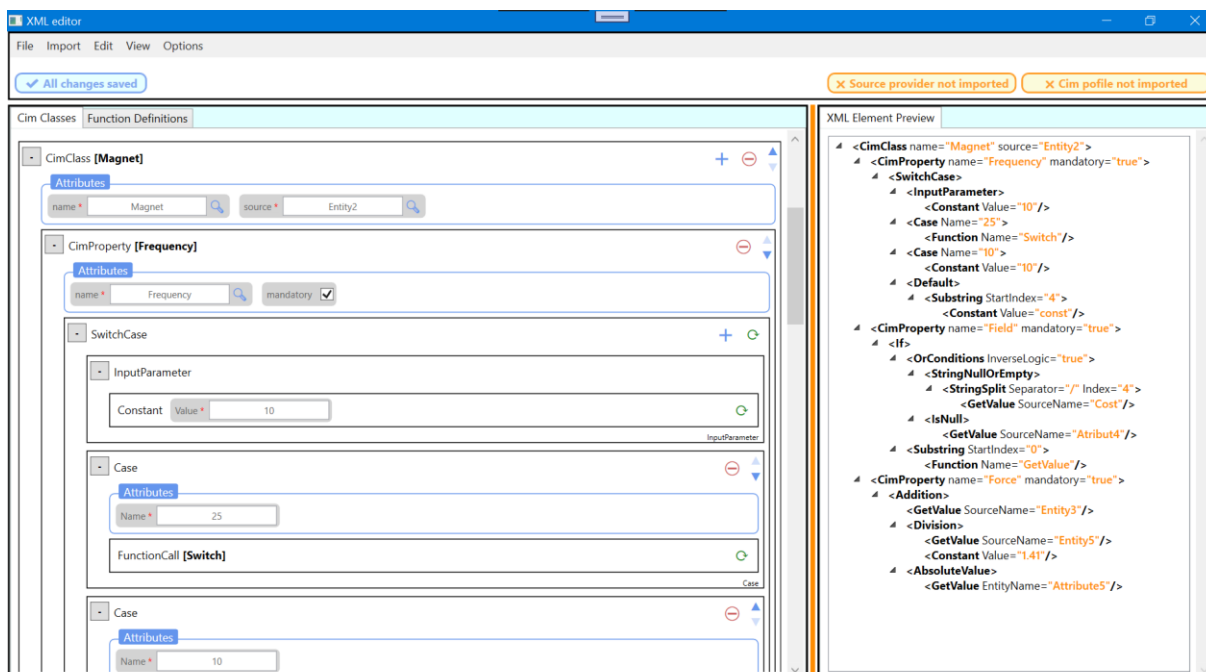
С леве стране главног прозора налази се радна површина намењена за приказ структуре документа. Како се документ састоји од секција намењених за дефиниције класа и функција, тако је и радна површина подељена у две картице – Cim Classes и Function Definitions. Десна страна главног прозора апликације намењена је за преглед XML репрезентације изабраних елемената у структури, а горњи део прозора садржи мени са опцијама које корисник има на располагању.

Почетно стање апликације подразумева празан документ спреман за уређивање и додавање елемената у њега, али корисник може и да отвори постојећи документ и да настави да рад на њему. Постојећи документ је могуће отворити коришћењем опције File -> Open (или путем пречице CTRL + O) након чега се отвара дијалог за избор XML датотеке из локалног фајл система. Након избора постојећег документа, на радној површини се приказује структура документа, као што је приказано на слици 5.



Слика 5 – Структура отвореног документа

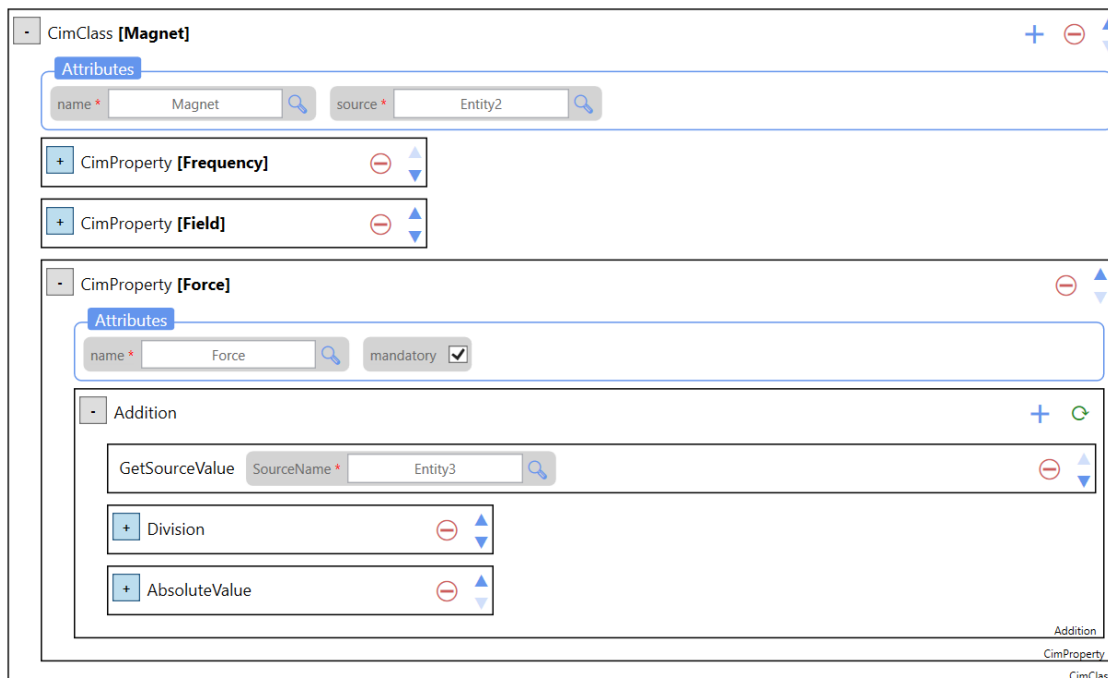
Кликом корисника на неки елемент у структури документа, на десној страни главног прозора се приказује преглед његове XML репрезентације, као што се може видети на слици 6.



Слика 6 – Приказ XML репрезентације одабраног елемента

Ради веће прегледности документа, обезбеђена је могућност сакривања садржаја појединачних елемената у структури. Корисник ово може постићи кликом на дугме „-“

које се налази са леве стране имена елемента, чиме се сви потомци изабраног елемента сакривају са радне површине. Поновни приказ садржаја елемента се врши притиском на исто дугме. На слици 7 се може видети како ова функционалност изгледа у конкретној употреби од стране корисника.



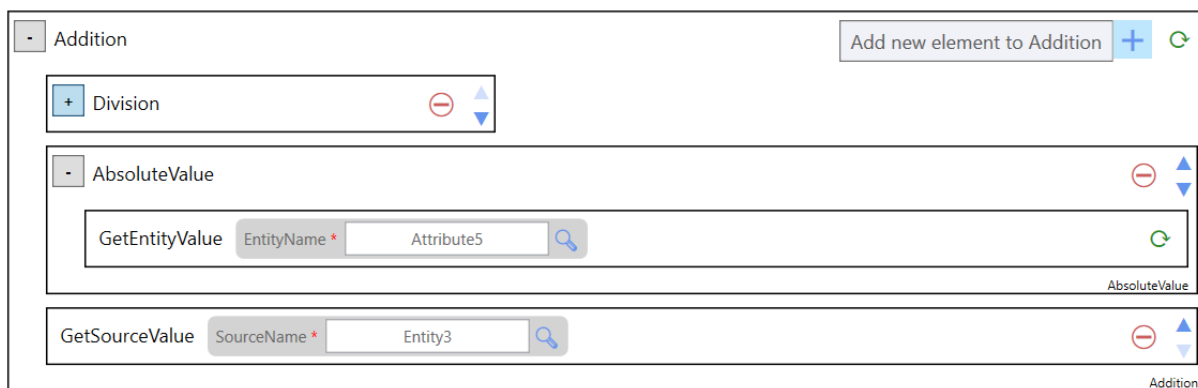
Слика 7 – Сакривање садржаја елемента

Статусна линија при врху главног прозора апликације садржи индикаторе о повезаности улазних датотека са апликацијом. Корисник улазне датотеке може увести у апликацију избором опција Import->Cim profile и Import->Source provider, након чега се подаци из ових датотека могу користити за мапирање на вредности атрибута елемената у документу.

## 6.2 Уређивање документа

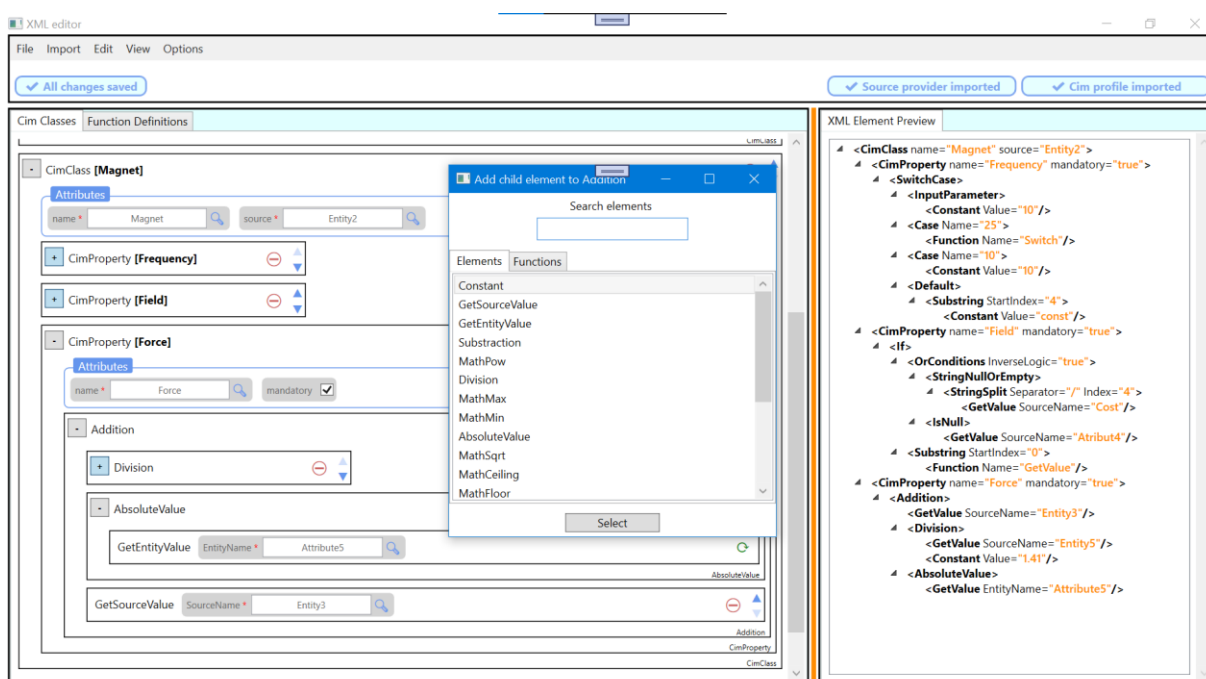
Након што је отворио документ који жели да уређује и увезао све улазне датотеке неопходне за мапирање података, корисник може да почне са уређивањем документа. Уређивање документа подразумева неколико операција над његовим елементима: додавање новог елемента, брисање елемента, замена елемента, замена места елемената, преименовање функције, избор вредности атрибута елемената.

Корисник нове елементе може додавати у садржај већ постојећих елемената, уколико је тај садржај проширив и има довољно простора за додавање новог елемента. Елементи у структури чији садржај испуњава наведене критеријуме у свом заглављу садрже дугме за додавање новог елемента, као што је приказано на слици 8.



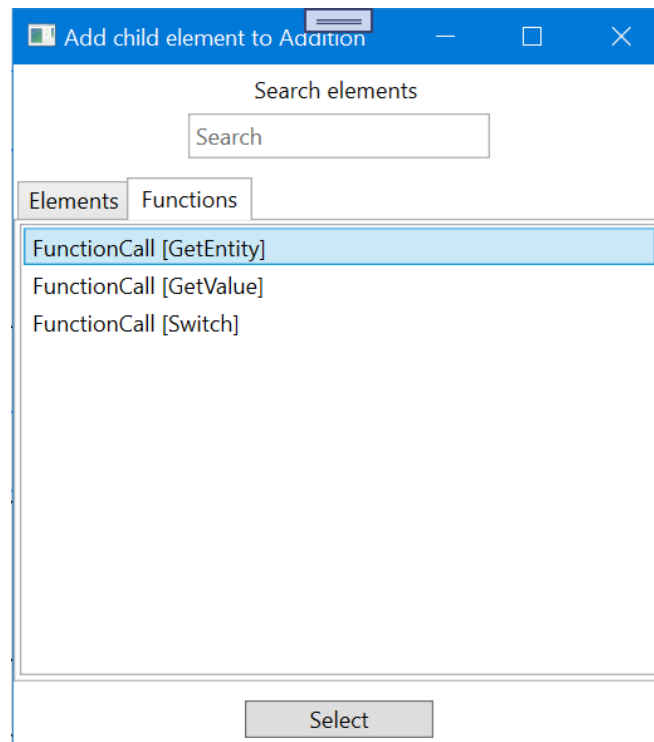
Слика 8 – Додавање новог елемента

Кликом на дугме за додавање новог елемента, кориснику се отвара дијалог за избор новог елемента. Кориснику ће бити понуђени само они елементи који одговарају правилима за попуњавање садржаја елемента у који се додају. У оквиру дијалога, кориснику је омогућено и да претражује елементе по њиховом имену, и тиме себи олакша избор. Изглед дијалога за избор новог елемента је приказан на слици 9.



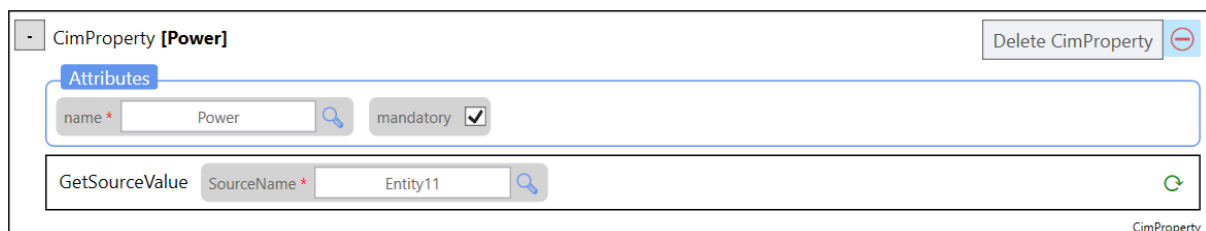
Слика 8 – Додавање новог елемента

Уколико родитељски елемент подржава позиве функција у свом садржају, елементи који представљају позиве функција такође могу бити изабрани у оквиру дијалога за додавање новог елемента. Корисник ове елементе може додати избором картице **Functions** у оквиру које му се приказују опције за позив функција које су дефинисане у секцији за дефиниције функција. На слици 10 је приказан овај случај коришћења.



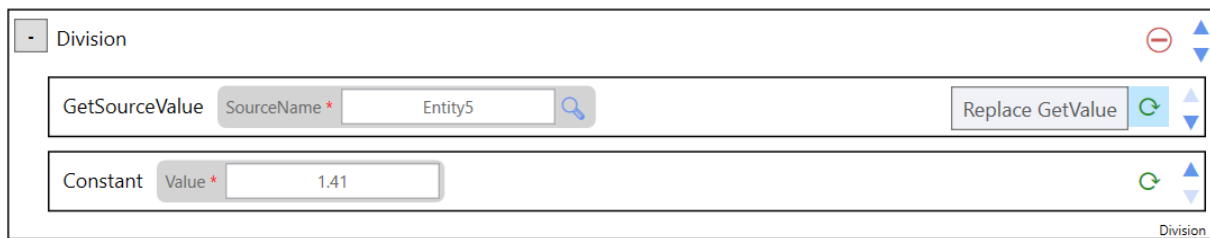
Слика 10 – Избор позива функције

Брисање елемената из структуре представља инверзну операцију процесу додавања елемената у документ. Елемент се може обрисати само уколико се његовим брисањем не би нарушила граматичка правила која важе у оквиру садржаја његовог родитељског елемента. Уколико испуњава овај критеријум, елемент у свом заглављу има приказано дугме за његово брисање, као што је приказано на слици 11. Брисањем елемента, заједно са њим се уклањају и сви његови потомци, тј. елементи у његовом садржају.



Слика 11 – Брисање елемента из структуре

Уколико елемент није могуће обрисати, кориснику се нуди опција да га замени са неким другим елементом, ако постоји елемент који може бити на његовом месту. Тада се уместо дугмета за брисање елемента кориснику приказује дугме за замену елемента, илустровано на слици 12. Кликом на ово дугме отвара се дијалог за додавање новог елемента, након чега корисник бира елемент којим ће заменити изабрани. Потом, елемент који се замењује се избацује из структуре заједно са свим својим потомцима, а нови елемент се додаје на његово место, чиме се поступак замене елемента завршава.



Слика 12 – Замена елемента

Кориснику је такође омогућено да мења места елементима у оквиру једног блока садржаја. Елементи који могу мењати места са суседним елементима у блоку садржаја са своје десне стране садрже стрелице које корисник може користити како би елементе померао за једну позицију навише или наниже. Употреба ових контрола је приказана на слици 13.



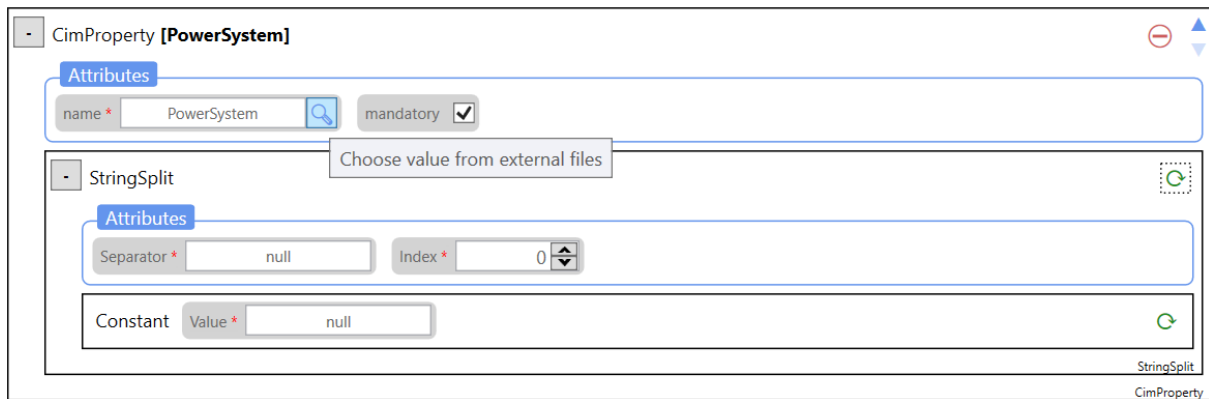
Слика 13 – Замена места елементима

Функције које је корисник дефинисао у сваком тренутку може преименовати. Елементи који представљају дефиниције функција у свом заглављу садрже и дугме за преименовање функције, као што је приказано на слици 14. Променом имена функције, мења се и вредност атрибута Name сваког елемента који се односи на позив те функције. Приликом промене имена функције се проверава да ли већ постоји функција дефинисана под тим именом, те се корисник спречава да направи две функције са истим именом, чиме се онемогућује двосмисленост документа.



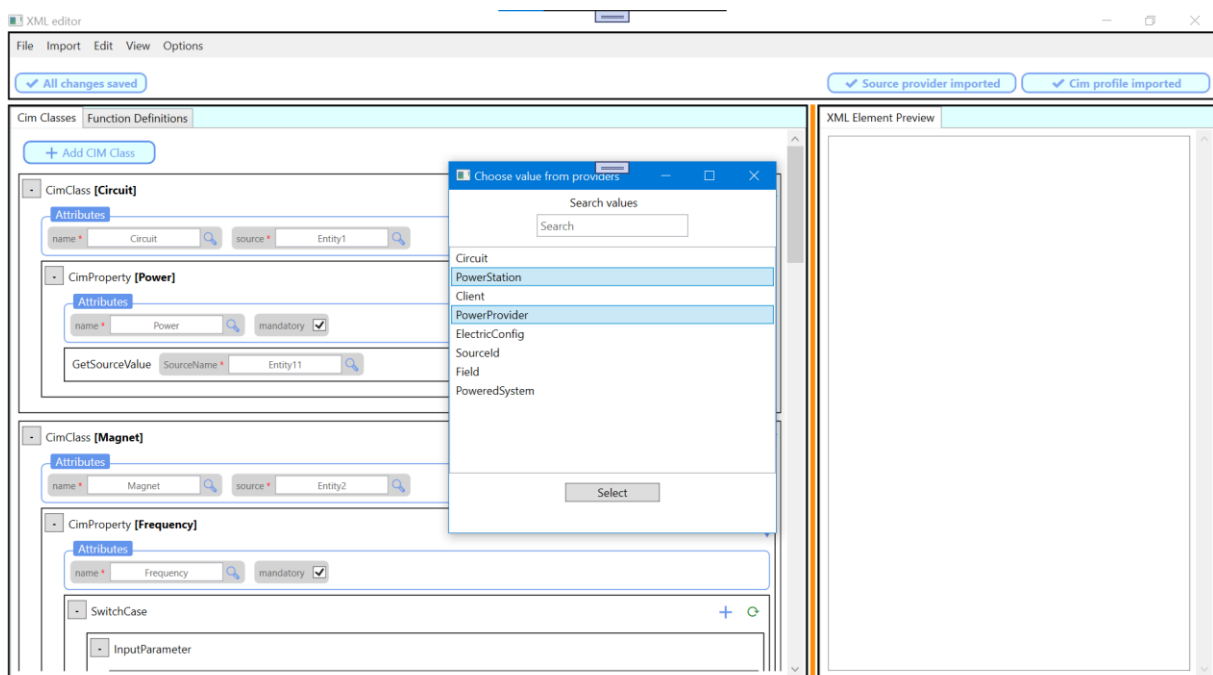
Слика 14 – Преименовање функције

Кориснику је омогућено да мења вредност атрибута сваког елемента у структури помоћу контрола корисничког интерфејса попут *textbox*, *checkbox* и *number input* контрола. Такође, атрибути чије вредности се мапирају на вредности из улазних датотека пружају и додатну опцију кориснику у виду дугмета на чији клик се отвара дијалог за избор вредности атрибута из улазних датотека. На слици 15 су приказани сви типови контрола и опција за избор вредности атрибута елемента.



Слика 15 – Измена вредности атрибута елемента

Дијалог за избор вредности атрибута из улазних датотека је приказан на слици 16. Из *Cim profile* и *Source provider* фајлова се учитава њихова структура и прикупљају потенцијалне вредности за одређене атрибуте елемената у документу. Отварањем овог дијалога, кориснику се приказују само оне вредности из улазних датотека које могу бити коришћене као вредности изабраног атрибута одређеног елемента, након чега корисник бира једну или више понуђених вредности.



Слика 16 – Дијалог за мапирање вредности атрибута



На крају свог рада на документу, корисник може изгенерисати излазни документ одабиром File->Save опције у менију (или пречицом CTRL + S), након чега из локалног фајл система бира у који излазни фајл жели да сачува документ. Уколико је на почетку коришћења апликације корисник рад почео на постојећем документу, његове измене ће подразумевано бити сачуване у истој датотеци из које је документ на почетку учитан. На слици 17. је дат пример излазног документа, изгенерисаног на основу његове еквивалентне структуре у оквиру апликације.

```

</> document.xml
1  <Root>
2    <FunctionDefinitions>
3      <Function Name="GetEntity">
4        <StringRemove StartIndex="3" Count="5">
5          <GetValue EntityName="Entity" />
6        </StringRemove>
7      </Function>
8      <Function Name="GetValue">
9        <If>
10         <Condition InverseLogic="false">
11           <IsNull>
12             <GetValue SourceName="Src" />
13           </IsNull>
14         </Condition>
15         <GetValue SourceName="Src" />
16       </If>
17     </Function>
18   </FunctionDefinitions>
19   <CimClasses>
20     <CimClass name="Circuit" source="Entity1">
21       <CimProperty name="Power" mandatory="true">
22         <GetValue SourceName="Entity1" />
23       </CimProperty>
24     </CimClass>
25     <CimClass name="Magnet" source="Entity2">
26       <CimProperty name="Frequency" mandatory="true">
27         <SwitchCase>
28           <InputParameter>
29             <Constant Value="10" />
30           </InputParameter>
31           <Case Name="25">
32             <Function Name="Switch" />
33           </Case>
34           <Case Name="10">
35             <Constant Value="10" />
36           </Case>
37           <Default>
38             <Substring StartIndex="4">
39               <Constant Value="const" />
40             </Substring>
41           </Default>
42         </SwitchCase>
43       </CimProperty>
44     </CimClass>
45   </CimClasses>
46 </Root>

```

Слика 17 – Пример изгенерисаног излазног документа

## 7. ЗАКЉУЧАК

У овом раду је представљено софтверско решење којим се олакшава уређивање и рад на XML документу, намењеном мапирању података електроенергетске мреже у СИМ модел. Развој апликације описан је кроз анализу спецификације захтева, али и спецификације дизајна која подразумева опис модела података и архитектуре система. Објашњена је имплементација решења у оквиру наведених коришћених технологија. На крају, представљен је ток интеракције корисника са графичким интерфејсом апликације у неколико најважнијих случајева коришћења.

Предности представљене апликације огледају се у прегледнијем приказу XML документа који се уређује, као и у интуитивном корисничком интерфејсу, једноставном за употребу. Како је граматика самог документа издвојена у XML шему, омогућена је лака проширивост или изменљивост исте, чиме се дозвољава флексибилност система са којима се ова апликација интегрише. Кориснику је омогућено да буде растерећен по питању познавања компликованих синтаксних и пословних правила документа који уређује, тако што му апликација у сваком моменту нуди избор могућих акција и самостално предлаже подразумевање. Овим се гарантује валидност документа у сваком моменту коришћења апликације, без простора за грешку од стране корисника.

Као потенцијални недостаци представљеног решења издвајају се недостатак подршке за дефинисање простора имена за појединачне елементе у документу, као и недостатак ефикасног начина претраге документа. Такође, кориснику би од велике помоћи биле опције за репликацију појединих делова документа, као и опција за поништавање претходно начињене акције, али те опције, за сада, нису имплементирани у оквиру представљеног решења.

У поређењу са постојећим сродним решењима, апликација представљена у овом раду свакако јесте прилагођенија документима намењеним специфичној употреби и специфичном домену, док су сродна решења општија и подржавају рад на свим типовима XML докумената, за сваку намену. Уска примена овој апликацији омогућава да буде једноставнија за употребу од бројних сродних решења, поготово за кориснике са мањим искуством у раду на XML документима. Међутим, конкурентске апликације нуде много шири спектар напредних могућности које корисницима додатно олакшавају рад, што управо јесте нешто што апликацији описаној у овом раду недостаје.

План за даљи развој апликације је у највећој мери фокусиран на побољшање корисничког искуства у виду богатијег избора акција и могућности са циљем лакшег и бржег рада на документу. Планирано је и увођење опција за персонализацију корисничког интерфејса попут избора светле или тамне теме, прилагођавање фонта и сл. Такође, подршка за рад са просторима имена у документу је један од захтева највишег приоритета у плану за даљи развој апликације.

## 8. ЛИТЕРАТУРА

- [1] IBM, "About IBM", доступно на: <https://www.ibm.com/about?lnk=flatitem> (посећено 14.07.2024.)
- [2] History of Information, " IBM Introduces the Generalized Markup Language (GML) and SGML -- Fundamental Building Blocks in the Development of Computerized Page Formatting", доступно на: <https://historyofinformation.com/detail.php?id=1665> (посећено 14.07.2024.)
- [3] Library of Congress, " Standard Generalized Markup Language (SGML). ISO 8879:1986", доступно на: <https://www.loc.gov/preservation/digital/formats/fdd/fdd000465.shtml> (посећено дана 14.07.2024.)
- [4] W3Schools, "Introduction to XML", доступно на: [https://www.w3schools.com/xml/xml\\_what.asp](https://www.w3schools.com/xml/xml_what.asp) (посећено 14.07.2024.)
- [5] W3Schools, "XML Schema Tutorial", доступно на: [https://www.w3schools.com/xml/schema\\_intro.asp](https://www.w3schools.com/xml/schema_intro.asp) (посећено 14.07.2024.)
- [6] TechTarget, "Common Information Model (CIM)", доступно на: <https://www.techtarget.com/searchstorage/definition/Common-Information-Model> (посећено 14.07.2024.)
- [7] Altova, "XML Editor: XMLSpy - Altova", доступно на: <https://www.altova.com/xmlspy-xml-editor> (посећено 15.07.2024.)
- [8] Altova, "Altova", доступно на: <https://www.altova.com/> (посећено 15.07.2024.)
- [9] JSON, "Introducing JSON", доступно на: <https://www.json.org/json-en.html> (посећено 15.07.2024.)
- [10] Liquid Technologies, "Liquid Studio", доступно на: <https://www.liquid-technologies.com/xml-studio> (посећено 15.07.2024.)
- [11] Liquid Technologies, "Liquid Technologies", доступно на: <https://www.liquid-technologies.com/> (посећено 15.07.2024.)
- [12] Liquid Technologies, "XML Editor", доступно на: <https://www.liquid-technologies.com/xml-editor> (посећено 15.07.2024.)
- [13] W3Schools, "Xpath Tutorial", доступно на: [https://www.w3schools.com/xml/xpath\\_intro.asp](https://www.w3schools.com/xml/xpath_intro.asp) (посећено 15.07.2024.)
- [14] Oxygen XML Editor, " Oxygen XML Editor ", доступно на: <https://www.oxygenxml.com/> (посећено 15.07.2024.)
- [15] Sync.ro, "Syncro Soft SRL", доступно на: <https://www.sync.ro/> (посећено 15.07.2024.)
- [16] XMLBlueprint, "XMLBlueprint: XML Editor", доступно на: <https://www.xmlblueprint.com/> (посећено 15.07.2024.)
- [17] Microsoft, "C# | Modern, open-source programming language for .NET", доступно на: <https://dotnet.microsoft.com/en-us/languages/csharp> (посећено 17.07.2024.)

- [18] Microsoft, ".NET Framework", доступно на: <https://dotnet.microsoft.com/en-us/> (посећено 17.07.2024.)
- [19] Learn Microsoft, "System.Xml Namespace", доступно на: <https://learn.microsoft.com/en-us/dotnet/api/system.xml?view=net-8.0> (посећено 17.07.2024.)
- [20] Microsoft, "Visual Studio", доступно на: <https://visualstudio.microsoft.com/> (посећено 17.07.2024.)
- [21] Learn Microsoft, "Windows Presentation Foundation documentation", доступно на: <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/?view=netdesktop-8.0> (посећено 17.07.2024.)
- [22] Learn Microsoft, "Model-View-ViewModel (MVVM)", доступно на: <https://learn.microsoft.com/en-us/dotnet/architecture/maui/mvvm> (посећено 24.07.2024.)
- [23] Learn Microsoft, "Data binding overview (WPF .NET)", доступно на: <https://learn.microsoft.com/en-us/dotnet/desktop/wpf/data/?view=netdesktop-8.0> (посећено 24.07.2024.)

## БИОГРАФИЈА

Душан Суђић је рођен 10.10.2001. у Новом Саду. Основно образовање је стекао у Бачкој Паланци, а средње у Новом Саду. Школске 2020/21. године се уписује на Факултет техничких наука на студијски програм Рачунарство и аутоматика. Положио је све испите предвиђене планом и програмом и стекао услов за одбрану дипломског рада.