НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»

Теплоенергетичний факультет

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено
Завідувач кафедри
_____ О.В. Коваль
(підпис) (ініціали, прізвище)
“___”_____2020р.

ДИПЛОМНА РОБОТА

на здобуття ступеня бакалавра

з напряму підготовки 121 Інженерія програмного забезпечення
на тему”Розробка тематичного реєстру для дослідження комп’ютерних мереж”

Виконав (-ла): студент (-ка) __4__курсу, групи__ТВ61__

Сєров Максим Денисович
(прізвище, ім’я, по батькові) (підпис)

Керівник _____Колумбет Вадим Петрович_____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Консультант _____ _____ _____
(назва розділу) (вчені ступінь та звання, прізвище, ініціали) (підпис)

Рецензент_____ _____ _____
(посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Засвідчую, що у цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

Національний технічний університет України

“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет теплоенергетичний

Кафедра автоматизації проектування енергетичних процесів і систем

Рівень вищої освіти перший рівень

Напрям підготовки: 121 Інженерія програмного забезпечення

Спеціалізація: Програмне забезпечення розподілених систем

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ О.В. Коваль
(підпис)
”___”_____2020р.

ЗАВДАННЯ

на дипломну роботу студенту

Сєров Максим Денисович
(прізвище, ім’я, по батькові)

1. Тема роботи ”Розробка онтологічного реєстру для дослідження комп’ютерних мереж”

керівник роботи ст.в. Колумбет Вадим Петрович
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”__”___ 202__р. № __

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи Для створення системи були використані засоби мови програмування C# та мови розмітки XAML, фреймворк .NET WPF та бібліотеки dotNetRdf, QuickGraph, GraphSharp.

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Побудувати онтологічну модель реєстру комп’ютерної мережі , описати технічні та програмні засоби. Створити програму, яка вміє зчитувати, створювати, редагувати та зберігати OWL файли. Розробити користувацький інтерфейс, який дозволяє візуалізувати схему комп’ютерної мережі у вигляді графу, взаємодіяти з кожним її елементом.

5. Перелік ілюстративного матеріалу
«», «Структура проекту», «Функціональна модель системи», «Існуючі програмні рішення», «Реалізація функцій», «Засоби програмної реалізації», «Приклади інтерфейсу користувача», «Висновки»

7. Дата видачі завдання ”29”____Січня__2020 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	29.01.2020	
2.	Вивчення та аналіз задачі	30.01.2020-25.02.2020	
3.	Розробка архітектури та загальної структури системи	25.02.2020-12.03.2020	
4.	Розробка структур окремих підсистем	13.03.2020-25.03.2020	
5.	Програмна реалізація системи	26.03.2020-10.05.2020	
6.	Оформлення пояснювальної записки	10.05.2020-17.05.2020	
7.	Захист програмного продукту	18.05.2020	
8.	Передзахист	09.06.2020	
9.	Захист	15.06.2020	

Студент _____ _____Сєров М.Д._____
(підпис) (прізвище та ініціали,)

Керівник роботи _____ _____Колумбет В.П._____
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Метою роботи було створення клієнтського програмного засобу для дослідження та редагування онтології предметної області.

Представлення комп’ютерної мережі у вигляді графу та подальша візуалізація графу у схематичному вигляді. Надання користувачу можливості редагувати список пристроїв вхідної комп’ютерної мережі, так і схему її представлення. Додаток, що вміє зчитувати, редагувати, створювати та зберігати змінені файли онтологічних моделей.

Записка містить 37 сторінок, 18 рисунків, 1 таблицю та 12 посиланнь.

ABSTRACT

The aim of the work was to create a client software for researching and editing the ontology of the subject area.

Representation of a computer network in the form of a graph and subsequent visualization of the graph in a schematic form. Enabling the user to edit the list of devices on the incoming computer network and the scheme of its presentation. An application that can read, edit, create and save modified ontology model files.

The note contains 37 pages, 18 figures, 1 table and 12 references.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

1) OWL (Web Ontology Language) — мова для опису онтологій. Вона дозволяє описувати класи та зв’язки між ними.
2) RDF (Resource Description Framework) — модель для представлення даних, та в особливості метаданих . Представляє твердження про ресурси у вигляді, придатному для машинної обробки
3) WPF (Windows Presentation Foundation) — платформа для розробки веб-система для побудови клієнтських додатків Windows.
4) C# - мова програмування.

ЗМІСТ

ВСТУП

На сьогоднішній день обсяг знань в галузі комп'ютерних мереж дуже великий, і це ускладнює людині аналіз цих даних.

Велика кількість знань, наявних в мережі організована за допомогою семантичної мережі. Онтологія називається ядром Semantic Web, так як

вона необхідна для розробки додатків семантичної павутини. Розробка онтології в різних областях довела свою ефективність в різних напрямках. Онтологія є всеосяжною і детальної формалізацією деякої області знань з

допомогою таксономії. Таксономія, в свою чергу, є базовою складовою онтології, яка визначає класи об'єктів і взаємодія між ними. Таксономія ілюструється з допомогою ієрархічної структури даних, яка містить класи об'єктів відносяться до області знань, їх зв'язку, правила і обмеження, прийняті в цій області. Онтологія є найбільш потужним і широко поширеним інструментом моделювання відносин між об'єктами різних предметних областей.

Онтологічні моделі можуть забезпечити роботу з ними для кількох людей одночасно, що спростить вивчення та аналіз предметної області

Додаток OntoNet пропонує користувачам відображення комп’ютерної мережі у вигляді графу, можливість зручного пошуку інформації щодо компонентів цієї мережі та взаємозв’язків між ними. Як приклад було розроблено онтологію реєстру для дослідження комп’ютерної мережі кафедри. Від користувачів додатку не потребується специфічних знань проектування онтологій. Користувач може за допомогою прикладного програмного інтерфейсу власноруч редагувати модель комп’ютерної мережі та одразу побачити резульат. Зміни відображаються на візуалізованому графі. При виникненні чи зникненні елементу комп’ютерної мережі, або зв’язку між елементами користувач побачить, що відповідні елементи та зв’язки зникли и на візуалізації.

Також впроваджені функції редагування ієрархічного дерева онтології(дерева класів та підкласів – понять предметної області) та створення порожньої онтологічної моделі, яку можна заповнити за допомогою користувацького інтерфейсу, після чого використовувати сторонніми програмами, що підтримують OWL-файли.

1. ПОСТАНОВКА ЗАДАЧІ

Метою розробки додатку є реалізація інформаційної системи для аналізу комп’ютерних мереж, яка вміє формувати, зчитувати, зберігати онтологічні моделі для можливості подальшого використання сторонніми програмами. До реєстру, що цілком описує поняття предметної області “комп’ютерна мережа ” входять:

- Обладнання комп’ютерних мереж - кінцеві пристрої та проміжні пристрої, що поєднують кінцеві пристрої в мережі
- Середовище передачі даних
- Технології побудови локальних комп’ютерних мереж
- Технології побудови розподілених комп’ютерних мереж
- Стеки протоколів комп’ютерних мереж
- Адресація в комп’ютерних мережах
- Маршрутизація в комп’ютерних мережах
- Класи та топології комп’ютерних мереж

Створення онтології локальної комп'ютерної мережі забезпечить можливість розглянути компоненти мережі, що в свою чергу, забезпечить знання, що відносяться до даної предметної області

Загальний процес розробки онтології включає наступні етапи:

- складання глосарію термінів (понять)
- точні визначення термінів
- природною мовою
- побудова дерев класифікації
- понять (ієрархії класів)
- визначення атрибутів класів і їх значень
- додавання примірників класів
- створення систем логічних висновків.

У додатку повинен бути зручний інтерфейс, який дозволить користувачу взаємодіяти з онтологічними моделями, вже наявними у системі та завантажувати власні. Система має візуалізувати схему комп’ютерної мережі у вигляді графу, вершинами якого являються елементи комп’ютерної мережі, а ребрами – зв’язки між ними. Користувач має можливість виділити окремий елемент на схемі, наприклад – комп’ютер, та побачити з якими елементами мережі він пов'язаний, яку має адресу та ім’я, продовжність з’єднання та тип кабелю.

АНАЛІЗ ОНТОЛОГІЙ КОМП'ЮТЕРНИХ МЕРЕЖ ТА ІСНУЮЧИХ ПРОГРАМНИХ СИСТЕМ

2.1. Онтології комп'ютерних мереж

Онтології, розроблені для забезпечення знань в ширшій предметної області як

комп'ютерні мережі, обмежені. Існує онтологія комп'ютерних мереж, розроблена в

освітніх цілях, яка досліджує поняття, такі підкласи як комунікації, додатки, стандарти і безпеку мережі, для використання в як навчальний посібник "The Development of Ontology-Based Course for Computer Networks" [Ling et al., 2008]. Основним недоліком існуючої системи є те, що відносини між гадки не проаналізовані належним чином. зв'язки типу «є» і «є частиною» використовується для всіх відносин, що робить онтологію слабкою. Можна відмітити роботу, присвячену онтології домашньої комп'ютерної мережі An Ontology Based Approach Towards A Universal Description Framework for Home Networks [Docherty, 2010]. Дана робота розглядає компоненти такої мережі, вимоги та умови комунікації між різними пристроями в мережі.

Також існує безліч робіт, присвячених оцінці складності комп'ютерних мереж. Вони розглядають комп'ютерні мережі з різних точок зору і з різними цілями. Дослідження, присвячене оцінці складності мереж Complex Networks' Analysis Using an Ontology-Based Approach [Becheru and Badica, 2014], автори представляють нову онтологію, яка дозволяє виконувати аналізи, які засновані на знаннях про складних мережах з різними мережевими атрибутів і метриками. Створення онтології локальної комп'ютерної мережі забезпечить можливість розглянути компоненти мережі, що в свою чергу, забезпечить знання, що відносяться до даної предметної області.

2.2. Мови опису онтологій RDF та OWL

Будучи основою для представлення даних, RDF використовується для побудови графіків знань - широко взаємопов'язаних, сумісних та гнучких інформаційних інфраструктур.

RDF - це загальний метод опису даних шляхом визначення зв’язків між об'єктами даних.

RDF дозволяє ефективно інтегрувати дані з різних джерел. Від'єднання даних від її схеми. Це дозволяє застосовувати декілька схем, взаємозв'язувати, запитувати як одну та змінювати, не змінюючи

RDF побудований на основі існуючих веб-стандартів: XML та URL (URI).

Мета RDF полягає в тому, щоб забезпечити стандартизований спосіб взаємозв'язку та доступу до всього, що ми знаємо та підтверджуємо на формальній, машинообробній мові.

RDFS (RDF Схема) - набір класів і властивостей для моделі подання знань RDF, що становить основу для опису онтологій з використанням розширеного RDF-словника для структури RDF-ресурсів. RDFS використовує кодування у вигляді RDF, тому що відносяться до RDF триплети можуть зберігатися, оброблятися і запитуватися подібно описами RDF-ресурсів, наприклад, за допомогою SPARQL.

The Ontology Web Language (OWL) - це набір мов розмітки, призначених для використання програмами, яким потрібно обробляти вміст інформації, а не просто представляти інформацію людині.

Онтологія OWL описує ієрархічну організацію ідей у домені таким чином, яким може аналізувати та розуміти програмне забезпечення. OWL має більше можливостей для вираження сенсу та семантики, ніж XML, RDF та RDFS, і, таким чином, OWL виходить за межі цих мов у своїй здатності представляти вміст, що інтерпретується в Інтернеті. OWL дозволяє додавати більше обмежень у поданні ваших знань. Він категоризує властивості (відносини) на об'єкти та властивості даних та дозволяє додавати обмеження на свої властивості.

2.3. Аналіз існуючих програмних систем

Створення онтологій – складний та довгий процес, що вимагає від користувача наявності специфічних знань, зокрема знання мови OWL. У цьому розділі надано широкий огляд деяких доступних редакторів та середовища, які можуть бути використані для побудови та взаємодії з онтологіями, а також надано короткий опис кожного з інструментів.

2.3.1 Інструмент WebOnto

Додаток WebOnto розроблений для перегляду, створення і редагування онтологій. Для моделювання онтологій він використовує мову OCML (Operational Conceptual Modeling Language).

Зображення інтерфейсу користувача наведено на рисунку 2.6

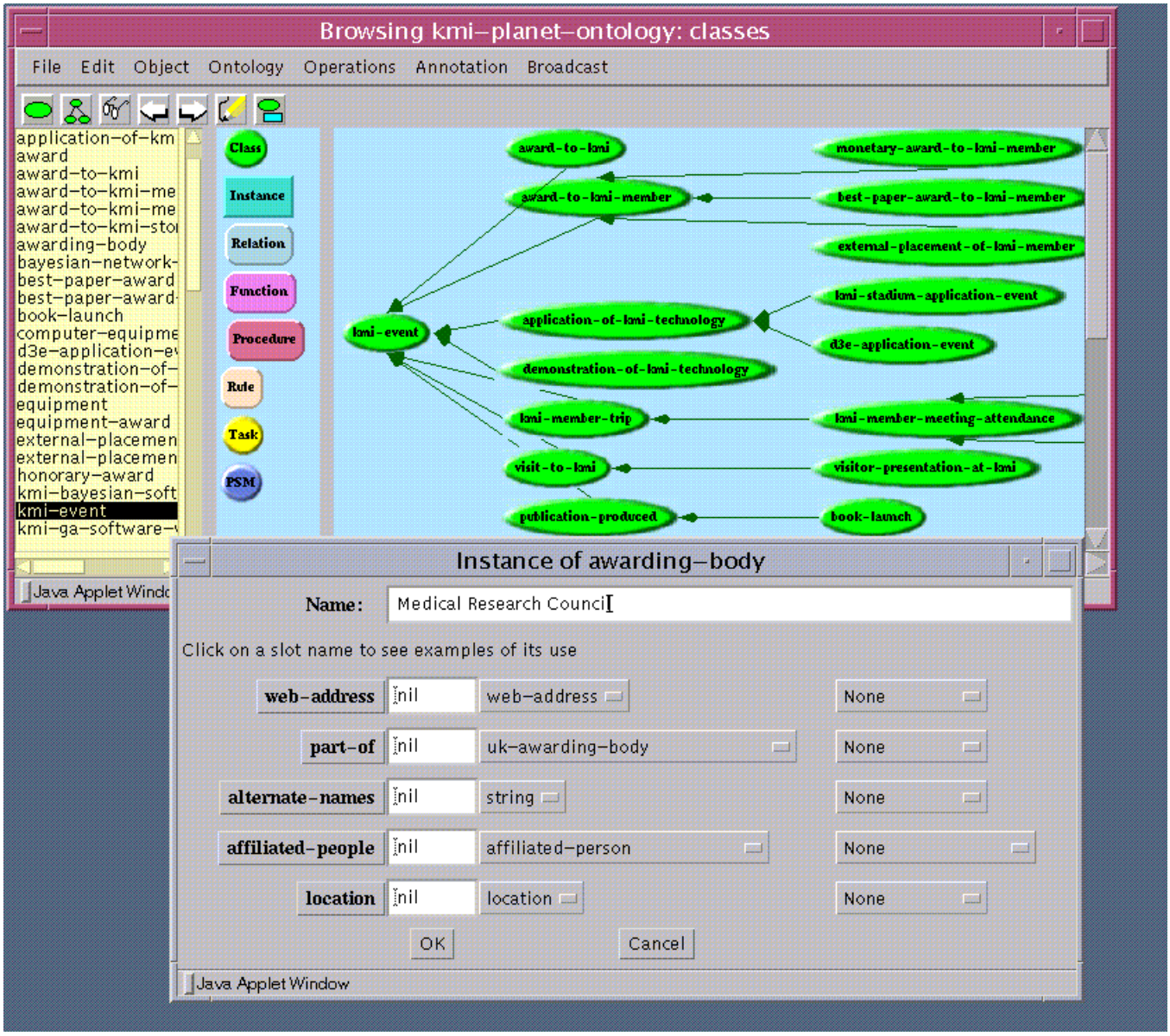

Рисунок 2.1 – Інтерфейс WebOnto

Користувач може створювати різні структури, в тому числі класи з множинним спадкуванням. Інструмент має ряд корисних особливостей: перегляд відносин, класів і правил, можлива спільна робота над онтологією декількох користувачів.

2.3.2 Інструмент Apollo

Apollo (The Open University, n.d.) - це зручна програма для моделювання знань. Apollo дозволяє користувачеві моделювати онтологію з базовими поняттями, такими як класи, екземпляри, функції, відносини тощо. Внутрішня модель - це віконна система, заснована на протоколі OKBC. База знань Apollo складається з ієрархічної організації онтологій. Онтології можуть бути успадковані від інших онтологій і можуть використовуватися як якби це були самостійні онтології. Кожна онтологія є онтологією за замовчуванням, яка включає все примітивне заняття. Кожен клас може створити ряд екземплярів, а екземпляр успадковує всі слоти класу. Кожен слот складається з набору фасетів. Інтерфейс системи наведено на рисунку 2.2.

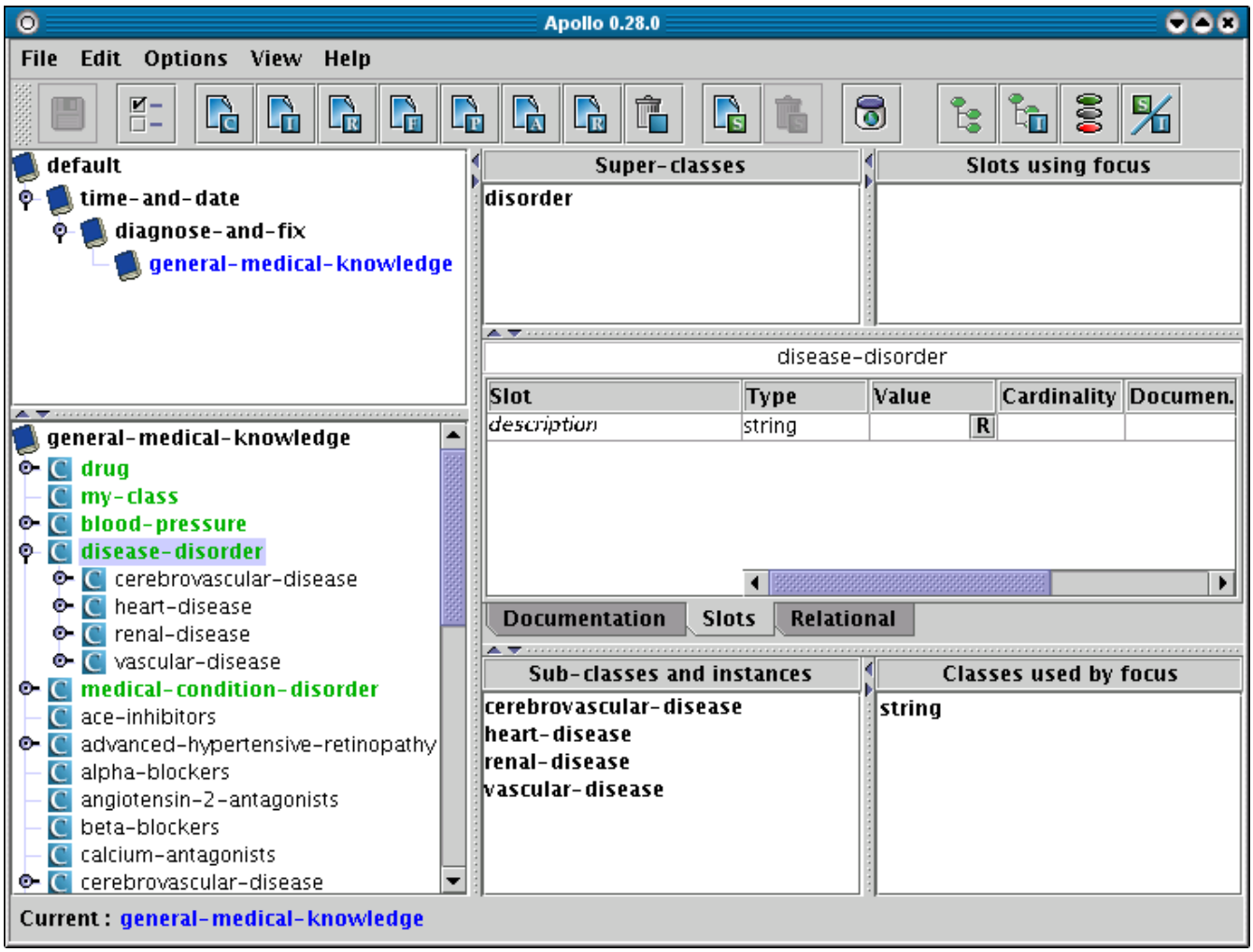

Рисунок 2.2 – Інтерфейс системи Apollo

Мінуси системи полягають в тому, що Apollo не підтримує перегляд графів, вилучення інформації та багатокористувацькі можливості, або спільну обробку. З переваг можна відмітити що Apollo відрізняється сильною перевіркою відповідності типів, збереження онтологій та імпорт / експорт форматів.

2.3.3 Середовище OntoStudio

OntoStudio Ontoprise розроблений на IBM Eclipse framework. Після завантаження користувач отримає доступ до трьохмісячного безкоштовного пробного періоду. Це середовище створення онтологій, у якому здійснюється розробка та підтримка онтологій з використанням графічних засобів. Він заснований на клієнт-серверній архітектурі, де онтологіями керує центральний сервер і різні клієнти можуть отримати доступ та вдосконалити ці онтології. Він підтримує багатомовну розробку, і модель знань пов'язана з frame based мовами (це технологія, що використовується для представлення знань у штучному інтелекті.). OntoStudio підтримує можливість спільної роботи над розробкою онтологій.

На рисунку 2.3 представлено інтерфейс програми

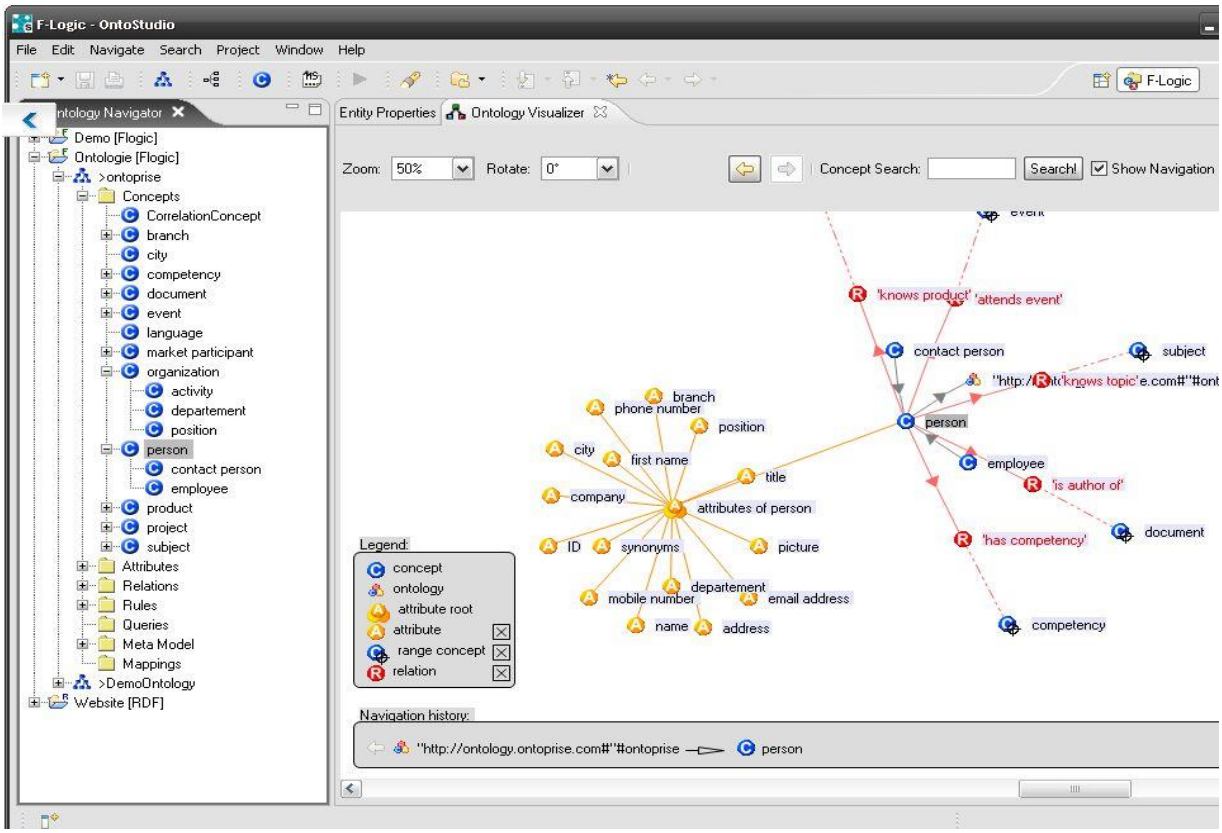

Рисунок 2.2 – Інтерфейс середовища OntoStudio

OntoStudio може імпортувати зовнішні дані у представленні DAML + OIL, Excel, F-логіці, RDF (S), схеми баз даних (Oracle, MS-SQL, DB2, MySQL) та OXML. OntoSudio також може імпортувати та експортувати файли OWL. OntoStudio надає API для доступу до онтологій у об’єктно-орієнтованому стилі. Завдяки модульній структурі OntoStudio може бути поповнена новостворюваними модулями і налаштована на потреби конкретного користувача.

З недоліків можна виділити не завжди інтуїтивно зрозумілий інтерфейс, платну повну версію програми. Створення запитів і екзмплярів класів відбувається у вигляді заповнення таблиць, на відміну від дружнього інтерфейсу середовища Protege, яке буде представлено у наступному розділі.

2.4 Висновки до розділу

Проведено огляд існуючих програмних рішень, редакторів онтологій для загального користування, та редакторів вузькоспеціалізованих. На основі аналізу було прийнято рішення щодо майбутньої розробки та програмних рішень. Перевагами програмного продукту повинні стати швидкість роботи програми, зручність взаємодії з графічним представленням саме моделей комп’ютерної мережі, коли

3.ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ПРОДУКТУ

За допомогою середовища Visual Studio 2015 мовою C# та з використанням сторонніх бібліотек було спроектовано та розроблено десктопний додаток OntoNet, у якому користувач може взаємодіяти з онтологіями. Але перш за все, онтологія комп'ютерної мережі була створена у середовищі Protégé.

3.1. Середовище розробки Visual Studio 2015

Переважна частина програмного коду системи написана на мові програмування C# із використанням фреймворку .NET та технології WPF, вибором середовища розробки став програмний продукт Visual Studio. Microsoft Visual Studio - це інтегроване середовище розробки від Microsoft. Воно використовується для розробки комп'ютерних програм, а також веб-сайтів, веб-додатків, веб-служб та мобільних додатків. Visual Studio використовує платформи Microsoft для розробки програмного забезпечення, такі як API Windows, Windows Forms, Foundation Presentation Foundation, Windows Store та Microsoft Silverlight. Visual Studio підтримує 36 різних мов програмування.

Для побудови графічного інтерфейсу системи була обрана технологія WPF (Windows Presentation Foundation). Якщо при створенні традиційних додатків на основі WinForms за отрисовку елементів управління і графіки відповідали такі частини ОС Windows, як User32 і GDI +, то додатки WPF засновані на DirectX. У цьому полягає ключова особливість рендеринга графіки в WPF: використовуючи WPF, значна частина роботи по відображенні графіки, як найпростіших кнопочок, так і складних 3D-моделей, лягати на графічний процесор на відеокарті, що також дозволяє скористатися апаратним прискоренням графіки.

На рисунку 3.1 схематично представлено архітектуру WPF:

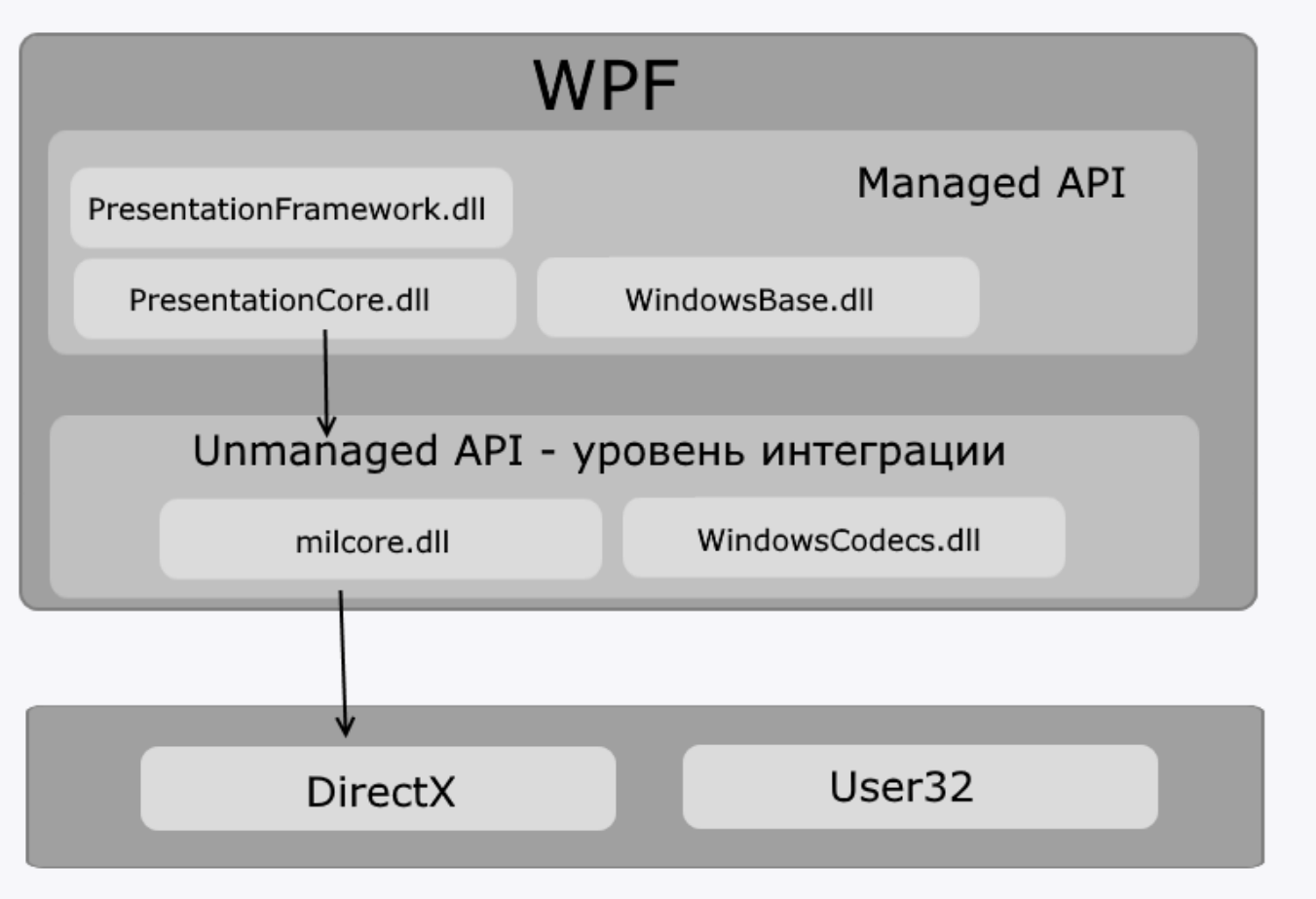

Рисунок 3.1 Схематичне представлення WPF

- Серед переваг WPF перед Windows Forms можна виділити наступні пункти: Декларативне програмування, web подібна модель компонування.
- Незалежність від роздільної здатності, апаратне прискорення, підтримка мультисенсорного введення.
- Стилі, тригери, шаблони, анімація, аудіо та відео, команди, потужна система прив'язки, маршрутизовані події.
- Просунуті можливості для малювання.
- Додатки з сторінкової організацією.

Зокрема як серйозну перевагу можна виділити поняття прив'язки Binding і DataContext, які радикально полегшують написання правильно структурованих програм, в яких уявлення відокремлено від моделі, бізнес-логіки і контенту.

Однією з важливих особливостей є використання мови декларативною розмітки інтерфейсу XAML, заснованого на XML: ви можете створювати насичений графічний інтерфейс, використовуючи або декларативне оголошення інтерфейсу, або код на керованих мовах C # і VB.NET, або поєднувати і те, і інше.

3.2. Платформа .NET Framework 4.5.2

Платформа .NET Framework - це керована середа виконання для ОС Windows, що надає різноманітні служби виконуваних у ній додатків. Вона складається з двох основних компонентів: середовища CLR - механізму, керуючого виконуються додатками, і бібліотеки класів .NET Framework - бібліотеки перевіреного коду, призначеного для повторного використання, який розробники можуть викликати зі своїх додатків. Ось які служби надає .NET Framework виконуваних у ній додатків. При розробці платформи .NET Framework враховувалися наступні цілі:

- Забезпечення узгодженої об'єктно-орієнтованого середовища програмування для локального збереження і виконання об'єктного коду, для локального виконання коду, розподіленого в Інтернеті, або для віддаленого виконання.
- Забезпечення мінімальних конфліктів при розгортанні програмного забезпечення та управлінні версіями.
- Забезпечення середовища виконання коду, що гарантує безпечне виконання коду.
- Відсутність проблем з продуктивністю середовищ виконання сценаріїв або інтерпретації коду.
- Забезпечення єдиних принципів розробки для різних типів додатків, таких як додатки Windows і веб-додатки.
- Взаємодія на основі промислових стандартів, яке гарантує інтеграцію коду платформи .NET Framework з будь-яким іншим кодом.

3.3. Бібліотека dotNetRDF

У середовищі Visual Studio 2015 відсутні вбудовані інструменти для роботи з файлами форматів RDF та OWL, тому для роботи я онтологіями було завантажено ефективне готове рішення від Microsoft, що має назву OwlDotNetAPI і входить до пакету dotNetRDF(рисунок 3.2). Завантажити це рішення до проекту можна за допомогою менеджеру пакетів Nuget.

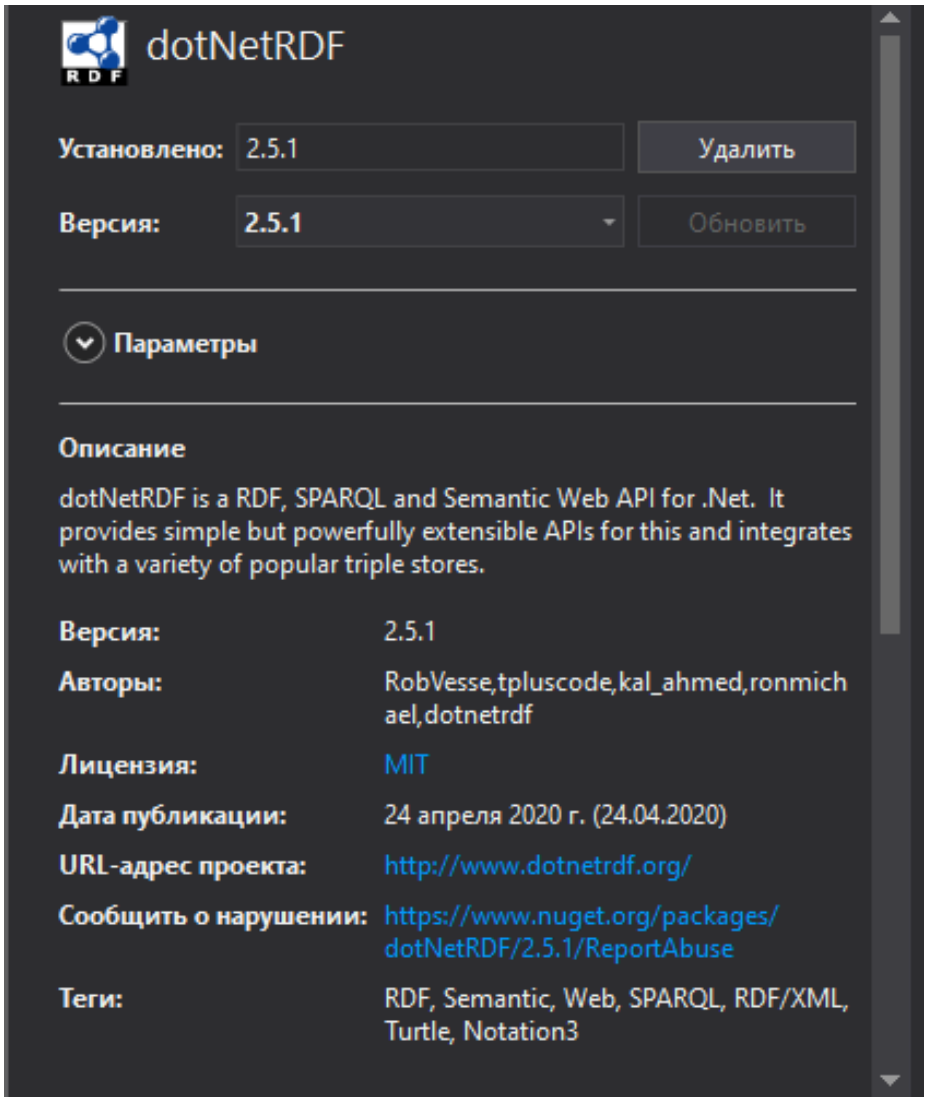

Рисунок 3.2 Бібліотека dotNetRDF

Бібліотека має методи для автоматичного парсингу онтологій у різних форматах, результат можна отримати у різних форматах, наприкад у вигляді графу до якого можна будувати та виконувати SPARQL-запити.

Засоби бібліотеки дозволяють оброблювати отриманий граф, змінювати його представлення відповідно до потрібного формату з яким зручно працювати.

3.4. Засіб опису онтологій Protégé

Одним з найбільш популярних засобів опису онтологій є система Protégé, розроблена в Стенфордському університеті (США). Це вільна, початково-відкрита платформа, яка забезпечує користувачів набором інструментів для створення і візуалізації онтологій в різних форматах уявлення, «метаінструмент», який

допомагає користувачам створювати системи придбання знань для конкретної предметної області, і експерти можуть використовувати ці системи для того,
щоб вводити і переглядати інформацію, що міститься в електронних базах знань. Модульна архітектура Protégé розширює клас систем, які можна збирати для виконання певних завдань у придбанні знань, і сприяє тому, що майбутні системи придбання знань можуть бути краще налаштовані відповідно до визначених
вимогами кінцевих користувачів.

Для розробки онтології потрібно створити класи, наповнити їх властивостями і екземплярами класів при додаванні властивості потрібно задати його ім'я і тип. Тип властивості може бути наступним: Class - властивість, успадковане від іншого класу; String - рядок; Double - дійсне число подвійної точності; Integer - ціле
число; Boolean - логічний тип даних. Якщо властивість має тип Class, то буде зберігатися примірник зі всі властивості успадкованого класу.

У Protege v3.4 для додавання примірника автоматично створюється форма введення, що містить всі властивості класу. Цю форму можна редагувати, розміщуючи властивості в зручному порядку. Така можливість дуже корисна для некваліфікованого користувача.

3.5. Бібліотеки QuikGraph та Graph#

представляє із себе розгалужену структуру, яку можна віднести до графу. Персональні комп'ютери, сервери, маршрутизатори та інші пристрої представляються як вузли графа, а зв'язки між ними - ребра графа, і саме це знання являється ключем до розв’язку задачі візуалізації мережі. Створення графу відбувається за допомогою бібліотек QuikGraph та Graph#. Перша забезпечує клас `BidirectionalGraph<object, IEdge<object>.`

Він служить посередником між бібліотеками Graph# та dotNetRdf.

Створення графу проілюстровано на рисунку 3.3:

```
// на основі файлу онтології створюємо граф, який будемо візуалізувати
private void CreateGraphToVisualize(ObservableCollection<INode> list)
{
    var g = new BidirectionalGraph<object, IEdge<object>>();


    string[] vertices = new string[50];
    for (int i = 0; i < list.Count; i++)
    {
        vertices[i] = ViewModel.SplitFunction(list[i].ToString());
        g.AddVertex(vertices[i]);
    }

    //додаємо вершини графу, кожна репрезентує пристрій у мережі

    foreach (string vert in vertices)
    {
        if (vert != null)
        {
            INode joined = OwlList.currentOwl.g.GetUriNode(":joined_with");
            INode comp = OwlList.currentOwl.g.GetUriNode(":" + vert);
            foreach (Triple s in OwlList.currentOwl.g.GetTriplesWithSubjectPredicate(comp, joined))
            {
                int e1 = Array.IndexOf(vertices, ViewModel.SplitFunction(comp.ToString()));
                int e2 = Array.IndexOf(vertices, ViewModel.SplitFunction(s.Object.ToString()));
                if (e2 >= 0)
                if (vertices[e1] != null && vertices[e2] != null)
                    g.AddEdge(new Edge<object>(vertices[e1], vertices[e2]));
            }
        }
    }
    GraphToVisualize = g;
}
```

Рисунок 3.3: програмна реалізація графу візуалізації

За допомогою бібліотеки dotNetRdf онтологічна модель представляється як список семантичних триплетів (суб’єкт, предикат, об’єкт). Для заповнення вершин графу обходимо список та шукаємо пристрої потрібного нам типу. Для побудови зв’язків шукаємо всі семантичні триплети, де об’єктом та суб’єктом є елементи комп’ютерної мережі, а предикатом є поняття що вказує на їх зв’язок.

4.ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

4.1 Створення онтології

Створення онтології складається зі створення ієрархії класів, встановлення зв’язків між класами, надання класам властивостей, створення екземплярів класів. Ієрархія класів комп’ютерної мережі зображена на рисунку 4.1, а її представлення у виді графу зроблене за допомогою інструменту OntoGrapf зображено на рисунку 4.2

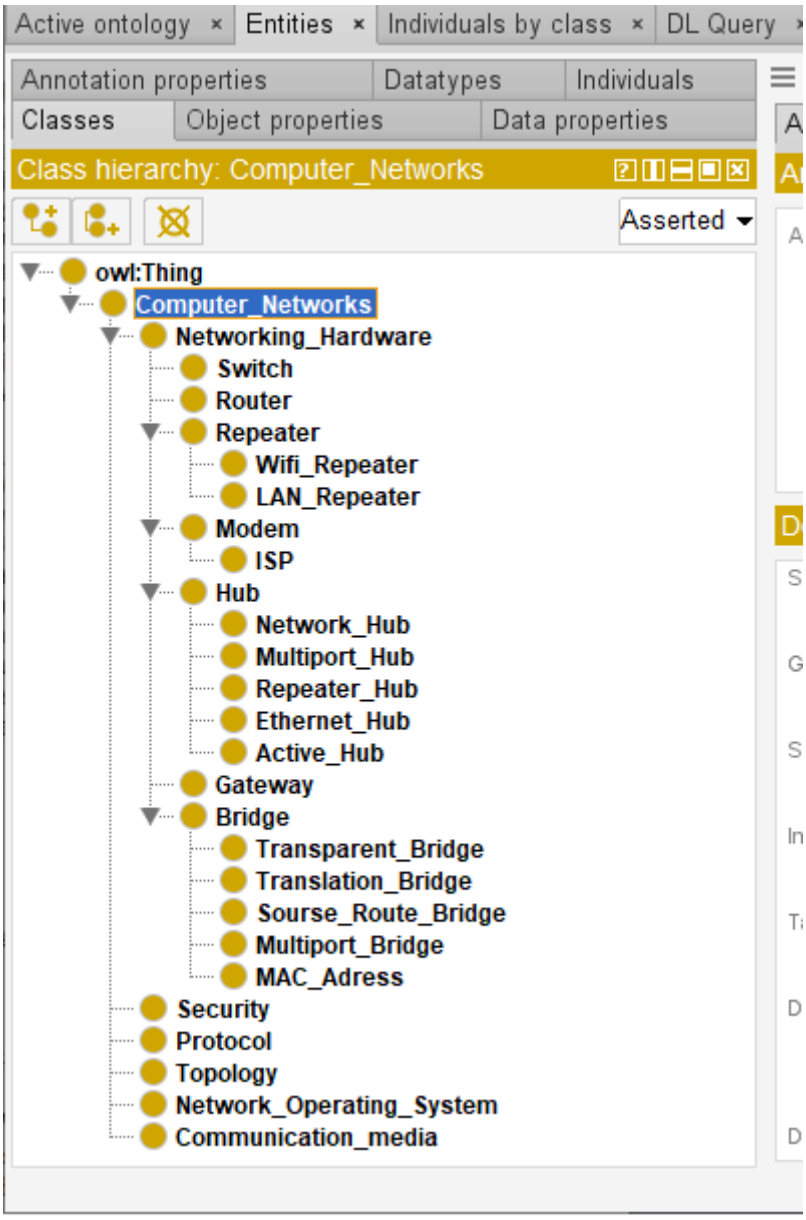

Рисунок 4.1 Ієрархія класів

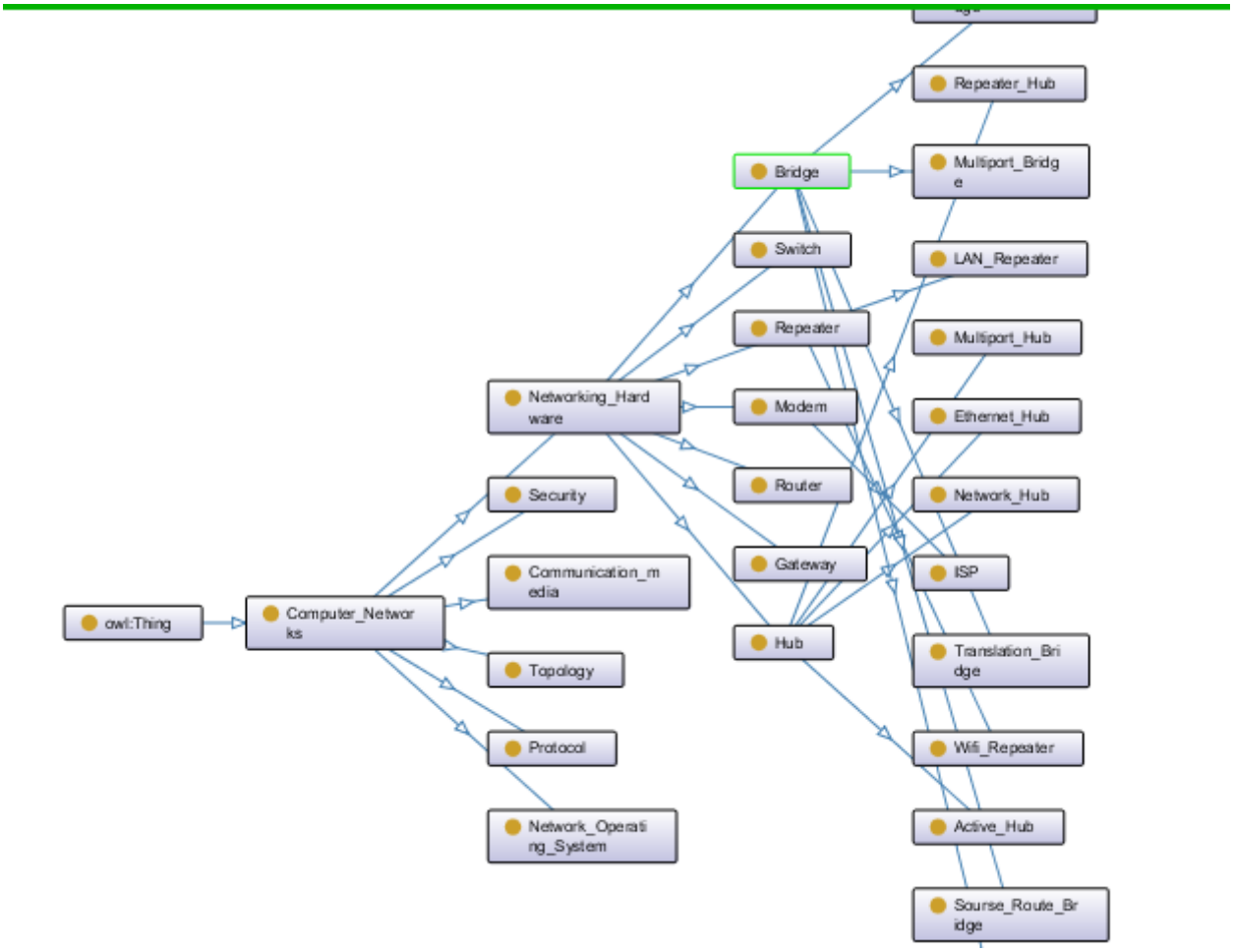

Рисунок 4.2 Зображення класів у вигляді графу

Після створення екземплярів потрібно задати властивості цих екземплярів, після чого потрібно заповнити атрибути для всіх класів.

При заповненні атрибутів для класу потрібно вказувати тип даних.

Зберігається створена онтологія в текстовому форматі на мові OWL або в форматі rdf.

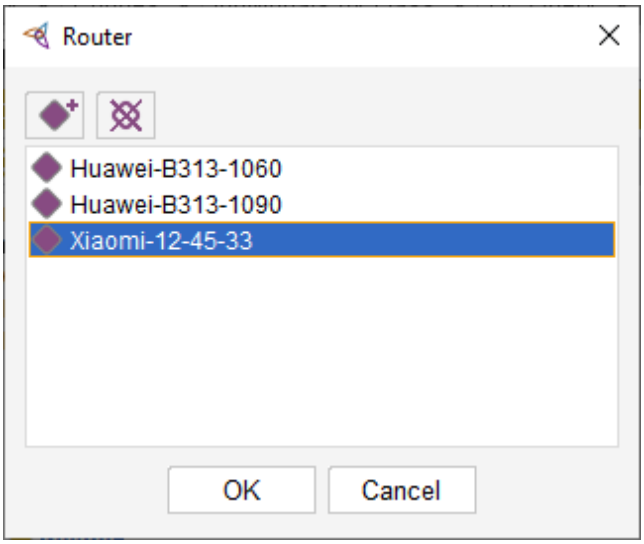

Рисунок 4.3 Список екземплярів класу Router

Створення екземплярів класів – завершуюча стадія створення онтології.

4.2 Створення додатку

Клієнтський додаток створено за допомогою платформи .NET Framework мовою C#. Бібліотеки QuickGraph та Graph# дозволяють візуалізувати граф.

Використання бібліотеки dotNetRDF забезпечує взаємодію зі створенною онтологією, в наявності методи зчитування та збереження файлів, представлення даних у зручному форматі. Зчитування моделі онтології можливе через завантаження файлів наступних форматів:

-OWL

-Rdf

-XAML

-txt

За допомогою інструментів що впроваджує бібліотека, онтологічна модель представляється у вигляді списків вузлів та семантичних триплетів, кожен з яких відповідно складається із трьох вузлів – суб’єкту, предикату і об’єкту, і декларує деяке твердження. З цих тверджень і складається онтологія. Наприклад, назва елементу комп’ютерної мережі , наприклад, персонального комп’ютеру з іменем “comp1”, буде репрезентована як вузол, а будь-яке твердження породжене через факт існування

цього комп’ютеру створює семантичний триплет, приклад продемонстровано на

рисунку 4.4:

Рисунок 4.4 – Список семантичних триплетів

У цьому списку перелічені твердження, у яких comp1 виступає суб’єктом. З них можна дізнатися, що:

-В рамках створеної онтології, comp1 це екземпляр класу;

-Цей екземпляр належить до класу кінцевих пристроїв EndDevices;

-Даний комп’ютер поєднується з двома іншими, comp2 та comp3;

-Представлена інформація щодо продовжності та типу кабелю, IP-адреси, встановленої операційної системи

Окрім зображених триплетів є ті, де comp1 предстає у ролі предиката чи об’єкта. Саме таке представлення даних надає можливість досліджувати предметну область, на взаємодії з триплетами побудована більша частина функцій програми, наприклад функція формування списку кінцевих пристроїв, наявних у мережі. Її сутність полягає в формуванні запиту до графу онтології, з ціллю повернути список всіх суб’єктів семантичних триплетів, де об’єкт – клас кінцевих пристроїв, а предикат – належність до класу.

Інший приклад – функція виводу у елемент управління TextBox інформації про всі характеристики елементу мережі, у якому зацікавлений користувач. Реалізація на рисунку 4.5:

```
//граф онтології, назва пристрою, поле у яке потрібно записати результат
public static void DataProperties(IGraph g, string dev, TextBox textBox)
{

    textBox.Text = "";
    List <INode> l = new List<INode>() ;
    INode n1 = g.GetUriNode(":" + dev);
    INode type = g.GetUriNode("rdf:type");
    INode dataProp = g.GetUriNode("owl:DatatypeProperty");

    foreach (Triple t in g.GetTriplesWithPredicateObject(type, dataProp))

        l.Add(t.Subject);


    }

    foreach (INode n in l)
    {
        foreach (Triple t1 in g.GetTriplesWithSubjectPredicate(n1,n))
        {
            textBox.Text +=SplitFunction(n.ToString()+": ");
            textBox.Text += SplitFunction2(t1.Object.ToString());
            textBox.Text += "\r\n" ;
        }


    }
```

Рисунок 4.5— Функція виводу інформації про елемент мережі

Таким самим чином реалізовані і інші функції, а саме завдяки можливості власноручно створювати онтологічні вузли та семантичні триплеті, додавати їх у існуючі онтологічні моделі. На рисунку 4.6 зображено програмну структуру

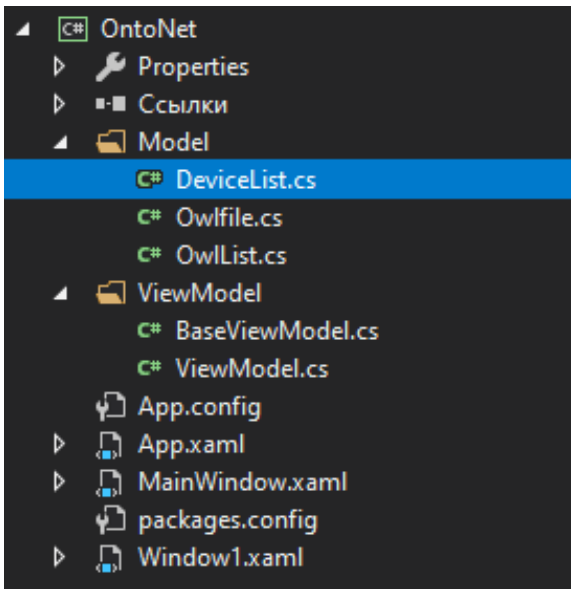

Рисунок 4.6-структура проекту

На рисунку 4.7 зображено узагальнену функціональну структуру системи:

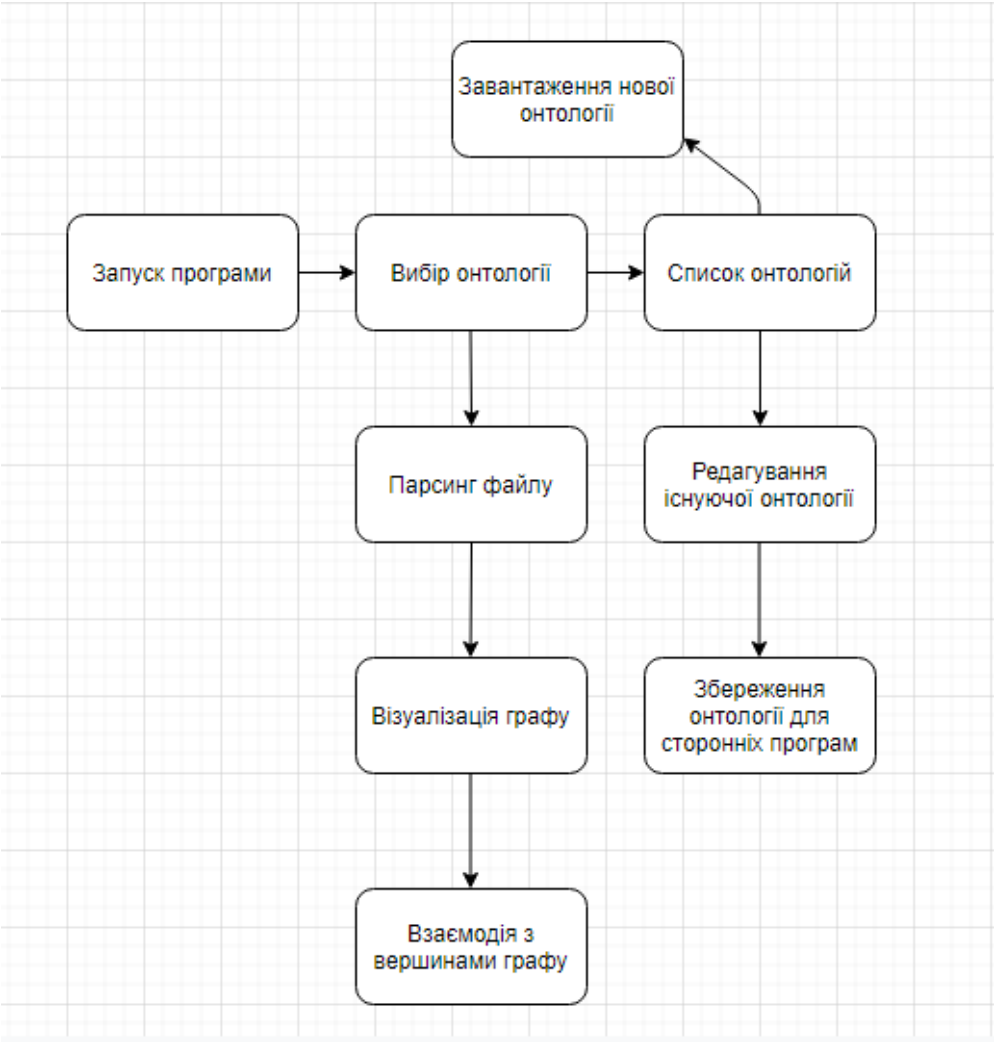

Рисунок 4.7 — Функціональна структура системи

За допомогою технології WPF, та мови розмітки XAML, що описує елементи управління та зовнішній вигляд елементів вікна, реалізовано зручний користувацький графічний інтерфейс.

5.МЕТОДИКА РОБОТИ КОРИСТУВАЧА З ПРОГРАМНОЮ СИСТЕМОЮ

5.1 Інсталяція та системні вимоги

Для запуску розробленої системи та роботи з нею не потрібна інсталяція, достатньо запустити виконуваний файл OntoNet.exe і робота одразу розпочнеться у з вікні програми що з’явиться.

Для встановлення розробленої програмної системи персональний комп’ютер

повинен мати процесор Intel ® Core ™ 2 / 2 Quad / Pentium ® / Celeron ® / Xeon™ чи AMD 6 / Turion ™ / Athlon ™ / Duron ™ / Sempron ™з тактовою частотою не нижче 2 GHz, на комп’ютері повинна бути встановлена операційна система Windows 10, Windows 8, Windows 7. Крім того, на персональному комп’ютері повинна бути встановлена бібліотека .NET версії 4.0 або новіше.

5.2. Сценарій роботи користувача з системою

Сценарій роботи користувача з системою простий, бо не потребує специфічних знань, потрібно просто запустити програму. Перед тим як починати аналізувати комп’ютерну систему, необхідно завантажити онтологію з комп’ютеру або обрати одну з тих що вже є у базі даних додатку. Для вибору файлу на персональному комп’ютері необхідно мати файл розширення .owl, натиснути кнопку завантаження файлу “File” --> “Open File” та вибрати його в директорії копм’ютеру(рисунок 5.1) Після цього з’являється можливість працювати з онтологією комп’ютерної мережі, дізнатися про наявні технічні та програмні засоби, організацію мережі. Окрім цього, можна обирати серед завантажених до середовища онтологій(рисунок 5.2), результат вибору буде виведен на екран. Одразу після завантаження проводиться аналіз та візуалізація

графу (рисунок 5.3). Отримуємо схематичне зображення конкретної комп’ютерної мережі з опцією масштабування, переміщення вершин графу у просторі, та переміщення у просторі графа цілком, утворивши саме таку схему яка буде зручною для користувача.

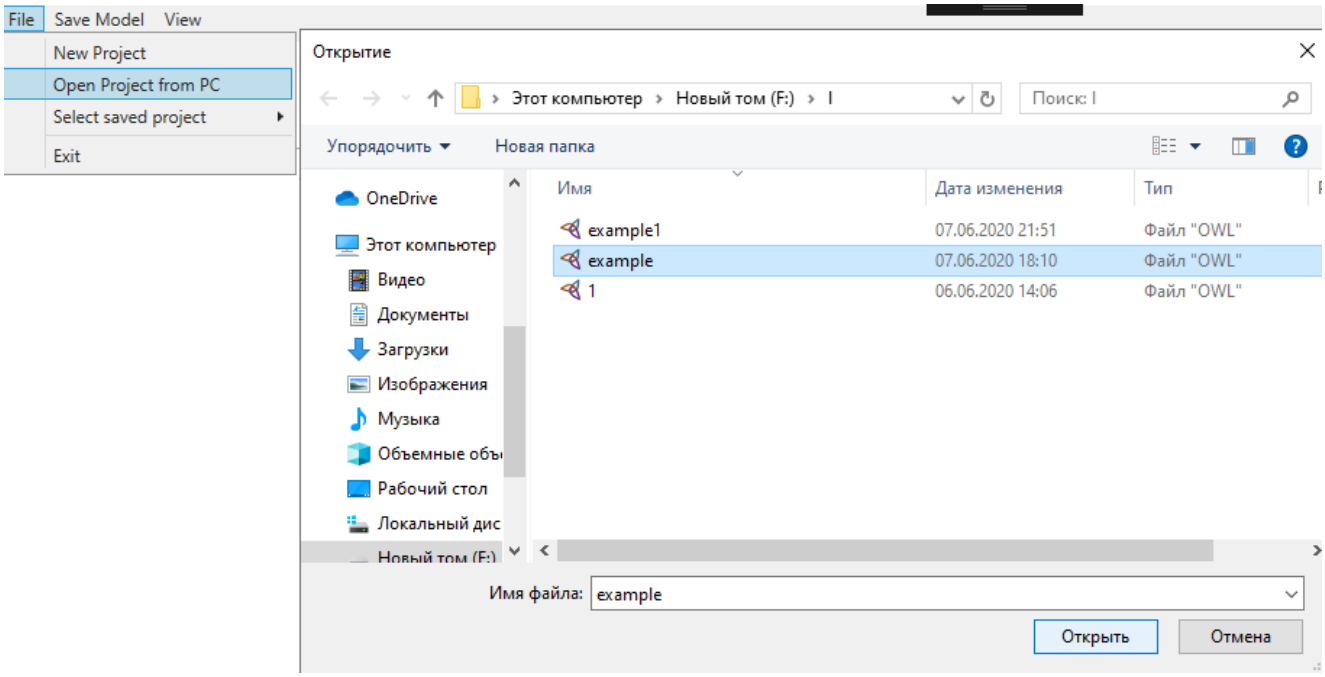

Рисунок 5.1 Кнопка вибору файлу на результат її роботи

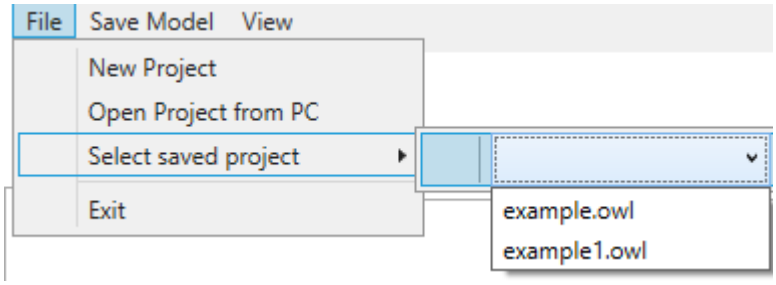

Рисунок 5.2 Вибір між вже завантаженими моделями

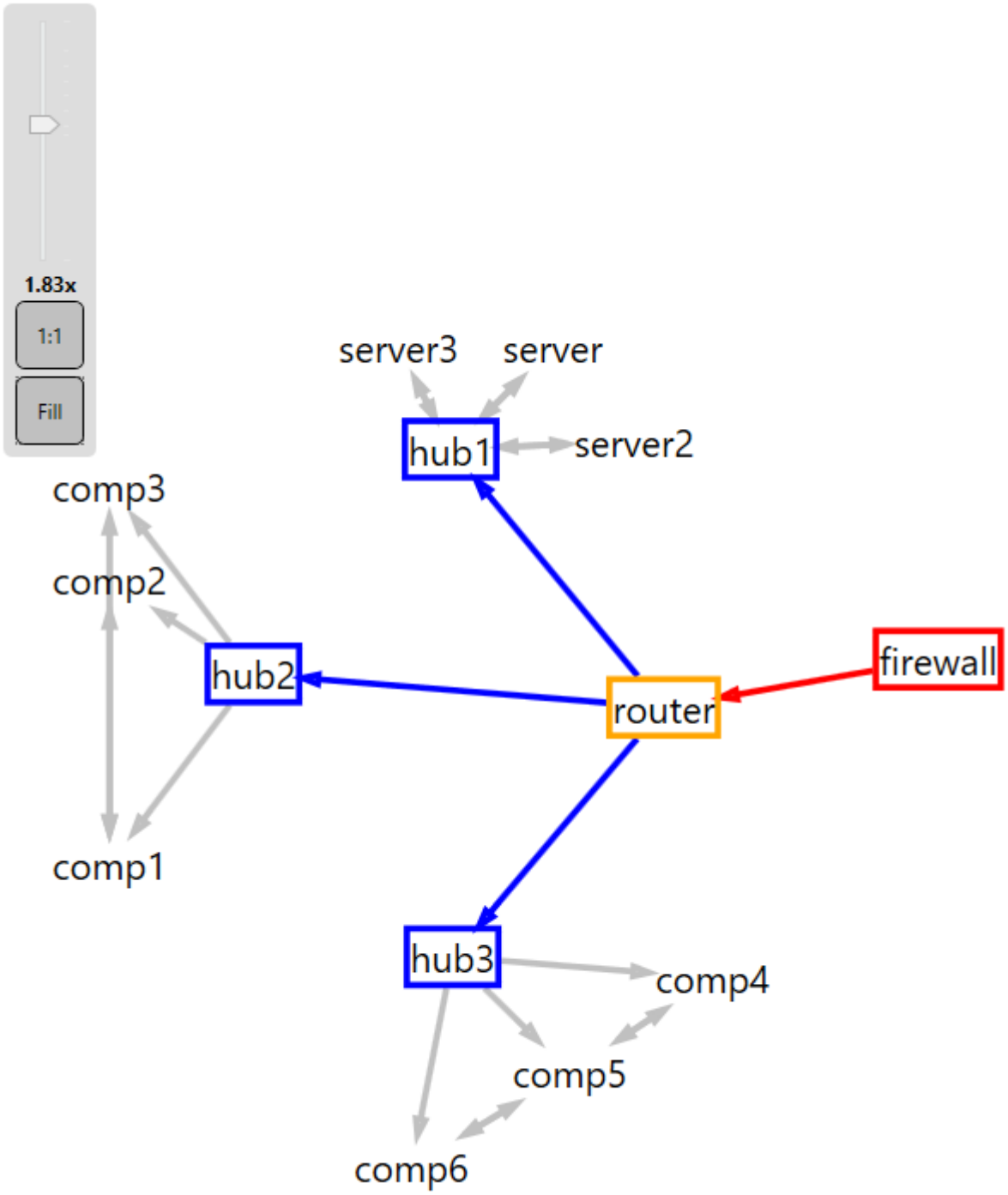

Рисунок 5.3 – Результат візуалізації мережі

Обравши пристрій мережі можна дізнатися з якими елементами мережі він пов'язаний, яку має адресу та ім’я, продовжність з’єднання та тип кабелю. Пристрій обирається в одному з двох списків, залежно від того який тип пристрію цікавить користувача. Результат вибору елементу продемонстровано на рисунку 5.4:

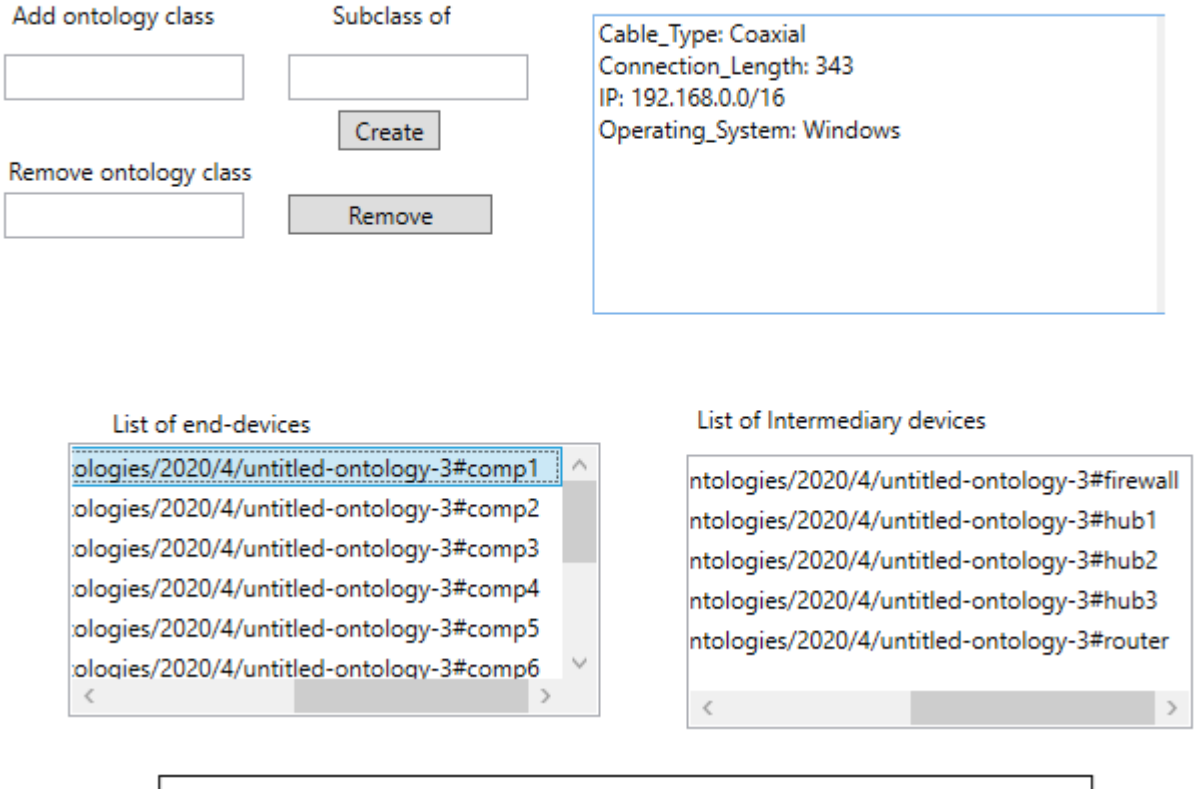

Рисунок 5.4 – Виведення інформації про пристрій

Крім того, користувач має можливість додавати нові класи та об’єкти до кожного з класів, що мають відношення до фізичних засобів мережі, збільшуючи наповнення онтології та зберігати ці онтології для подальшого використання сторонніми програмами. Інтерфейс має кнопки для додавання нових елементів мережі, задання характеристик для нових пристроїв, а також для видалення існуючих. Окрім цього, є можливість редагувати зв’язки між пристроями мережі, видаляти старі та встановлювати нові. На рисунку 5.5 зображено вікно збереження файлу.

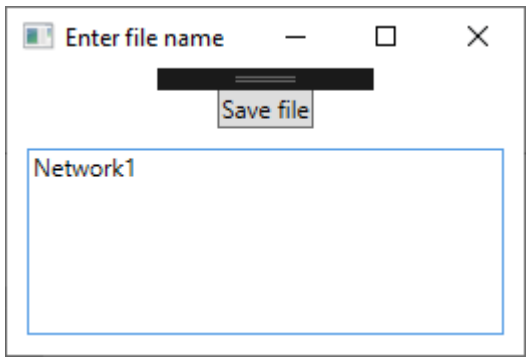

Рисунок 5.5 – Збереження редагованого файлу

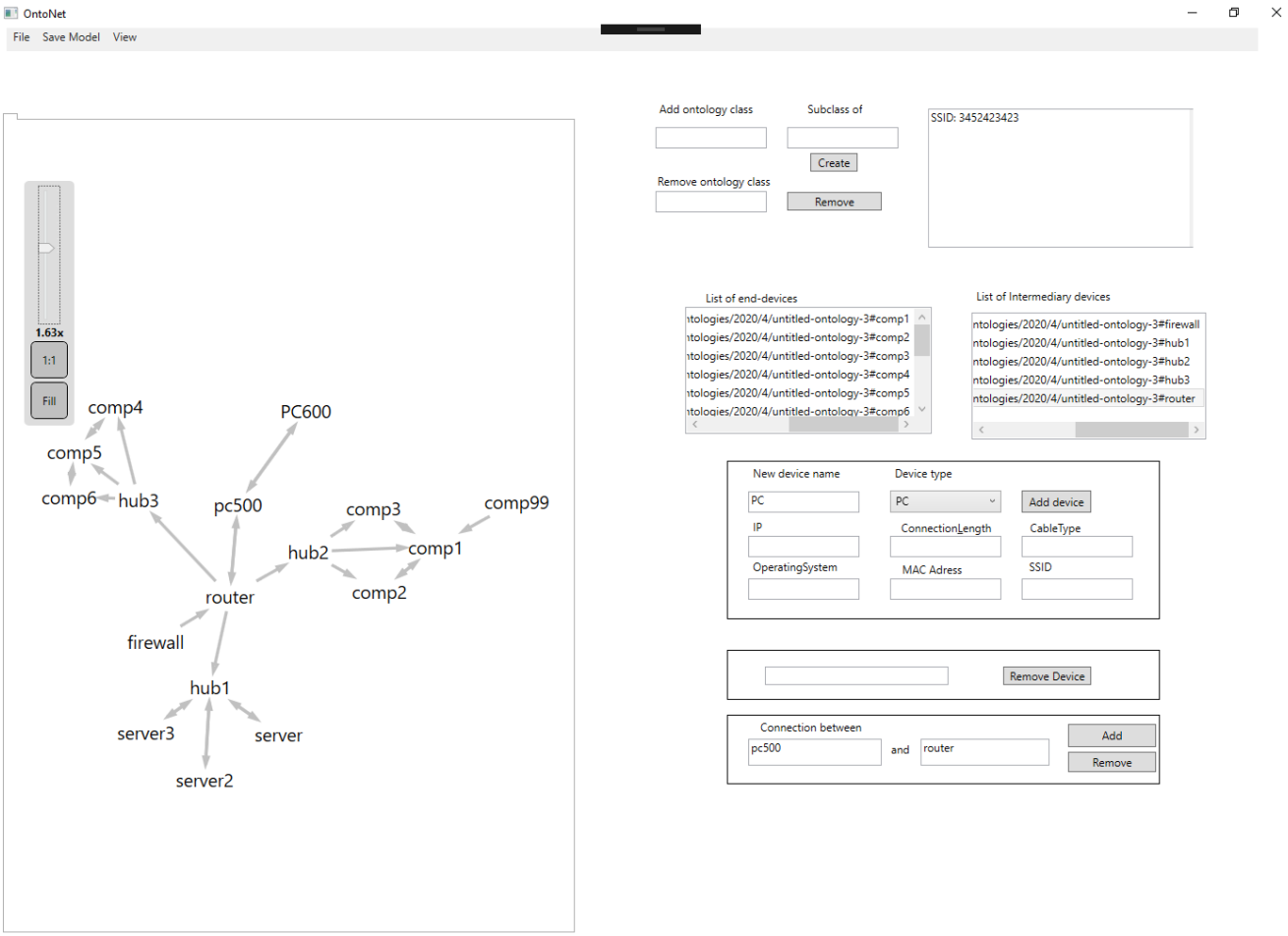

Рисунок 5.6 повний інтейфейс користувача

На рисунку 5.6 зображен програмний інтерфейс, що містить інструменти для реалізації всіх вищеназваних можливостей.

ВИСНОВКИ

Під час виконання дипломної роботи були виконані поставлені задачі.

1. Досліджено предметну область, створено базу знань яка містить у собі формалізоване представлення про комп’ютерні мережі, описано та поділено на класи найважливіші поняття що зустрічаються у предметній області, встановлені зв’язки між ними та впроваджено базу даних для екземплярів класів, що виступають прикладом комп’ютерної мережі.

2. Проведено огляд існуючих програмних рішень, редакторів онтологій для загального користування, та редакторів вузькоспеціалізованих. На основі аналізу було вирішено який функціонал необхідно впровадити в програмне рішення.

3. Додаток було вирішено створити за допомогою середовища розробки Microsoft Visual Studio та мови програмування C#, через існування технології WPF для зручного створення десктопних додатків зі зручним користувацьким інтерфейсом, та наявністю бібліотеки dotNetRdf, завдяки якій з’являється можливість парсингу RDF та OWL файлів. Для візуалізації графів використано інструмент GraphSharp.

4. Додаток дозволяє зчитувати, створювати, редагувати та зберігати онтологічні моделі. Впроваджено можливість візуалізації комп’ютерної мережі та дослідження конкретних її елементів, з можливістю взаємодії з обраним елементом. Можливість редагування онтологічної моделі, зокрема списку пристроїв комп’ютерної мережі, збереження багатьох моделей та швидке переключення між ними.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Ling Jiang, Chengling Zhao and Haimei Wei, "The Development of Ontology-Based Course for Computer Networks", International Conference on Computer Science and Software Engineering, 2008

2. Онтология в компьютерных системах [Електронний ресурс] — Режим доступу: https://rsdn.org/article/philosophy/what-is-onto.xml.

3. Horrocks I. From SHIQ and RDF to OWL: The Making of a Web Ontology Language / I. Horrocks, P. Patel-Schneider, F. van Harmelen.

4. SPARQL 1.1 Query Language, S. Harris, A. Seaborne, Editors, W3C Recommendation [Електронний ресурс] – 21 March 2013. — Режим доступу: http://www.w3.org/TR/sparql11-query.

5. [Docherty, 2010] Liam S. Docherty, An Ontology Based Approach Towards A Universal Description Framework for Home Networks, Technical Report CSM-182, Department of Computing Science and Mathematics, University of Stirling, 2010

6. [Becheru and Badica, 2014] Alex Becheru and Costin Badica, Complex Networks' Analysis Using an Ontology-Based Approach: Initial Steps, Lecture Notes in Artificial Intelligence 8793, pp. 326–337, Springer International Publishing Switzerland, 2014.

7. Рихтер Д. CLR via C# Программирование на платформе Microsoft .NET Framework 2.0 на языке C# / Джеффри Рихтер. – СПб: Питер, 2008. – 656 с. – (2).

8. Троелсен Э. Язык программирования C# и платформа .NET 4.5 / Эндрю Троелсен. – Киев: Вильямс, 2014. – 1312 с. – (6).

9. Semantic Web - W3C [Електронний ресурс] – Режим доступу до ресурсу: https://www.w3.org/standards/semanticweb/.

10. Gruber T. R. Towards principles for the design of ontologies used for knowledge sharing / T. R. Gruber. // Formal Ontology in Conceptual Analysis and Knowledge Representation. – 1993.

11. Visual C# 2008: базовый курс / [К. Уотсон, К. Нейгел, Я. Педерсен та ін.]. – М.: И.Д. Вильямс, 2009. – 1216 с.

12. Y. Sure. OntoEdit: Collaborative ontology development for the Semantic Web. Y. Sure, M. Erdmann, J. Angele, S. Staab, R. Studer, D. Wenke // In Proc. of the Inter. Semantic Web Conference (ISWC 2002), Sardinia, Italia, June 2002.

ДОДАТОК 1

Створення тематичного реєстру та візуалізація комп’ютерних мереж

Специфікація

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТВ6151_20Б

Аркушів 2

Київ – 2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ” КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС _ТВ6151_20Б 81-1	Записка.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ” КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС _ТВ51153_19Б 12-1	/Controllers/*.cs /Models/*.cs /Services/*.cs	Модулі клієнтської частини програмного продукту
УКР.НТУУ” КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС _ТВ6151_20Б 12-2	modules/workspace modules/subject modules/property modules.modals modules/result modules/config modules/i18n	Модулі клієнтського інтерфейсу програмного продукту
УКР.НТУУ” КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС _ТВ6151_20Б 13-1	Опис.docx	Опис модулю інтерфейсу клієнтської частини

ДОДАТОК 2

Створення тематичного реєстру та візуалізація комп’ютерних мереж

Лістинг програмного модулю

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТВ6151_20Б 12-1

Аркушів 10

Київ – 2020

```
using System;
using System.ComponentModel;
using System.Runtime.CompilerServices;
using VDS.RDF;
using VDS.RDF.Writing;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Collections.ObjectModel;
using System.Windows.Media.Imaging;
using QuickGraph;
using QuickGraph.Graphviz;
using System.IO;
using Microsoft.Win32;
using OntoNet.Model;

namespace OntoNet
{
   /// Логіка взаємодії для MainWindow.xaml

   public partial class MainWindow : Window, INotifyPropertyChanged
   {
      private IBidirectionalGraph<object, IEdge<object>> _graphToVisualize;

      public IBidirectionalGraph<object, IEdge<object>> GraphToVisualize
      {
         get { return this._graphToVisualize; }
         set
         {
            if (!Equals(value, this._graphToVisualize))
            {
               this._graphToVisualize = value;
               this.RaisePropChanged("GraphToVisualize");
            }

         }
      }

      public event PropertyChangedEventHandler PropertyChanged;

      public void RaisePropChanged(string name)
      {
         var eh = this.PropertyChanged;
         if (eh != null)
         {
            eh(this, new PropertyChangedEventArgs(name));
         }
      }



      public MainWindow()
      {
         InitializeComponent();
         // Контекст даних, посередник між представленням та моделлю даних
         OwlList comboBoxDataContext = new OwlList();
         DeviceList listDataContext = new DeviceList();

         listBox1.ItemsSource = DeviceList.getCompList();
         listBox2.ItemsSource = DeviceList.getconnectionDeviceList();
         comboBox1.ItemsSource = OwlList.getList();
         //  this.DataContext = new ViewModel();
```

```
    }

// на основі файлу онтології створюємо граф, який будемо візуалізувати
private void CreateGraphToVisualize(ObservableCollection<INode> list)
{
    var g = new BidirectionalGraph<object, IEdge<object>>();

    //add the vertices to the graph
    string[] vertices = new string[50];
    for (int i = 0; i < list.Count; i++)
    {
        vertices[i] = ViewModel.SplitFunction(list[i].ToString());
        g.AddVertex(vertices[i]);
    }

    //додаємо вершини графу, кожна репрезентує пристрій у мережі

    foreach (string vert in vertices)
    {
        if (vert != null)
        {
            INode joined = OwlList.currentOwl.g.GetUriNode(":joined_with");
            INode comp = OwlList.currentOwl.g.GetUriNode(":" + vert);
            foreach (Triple s in OwlList.currentOwl.g.GetTriplesWithSubjectPredicate(comp, joined))
            {
                int e1 = Array.IndexOf(vertices, ViewModel.SplitFunction(comp.ToString()));
                int e2 = Array.IndexOf(vertices, ViewModel.SplitFunction(s.Object.ToString()));
                if (e2 >= 0)
                if (vertices[e1] != null && vertices[e2] != null)
                    g.AddEdge(new Edge<object>(vertices[e1], vertices[e2]));
            }
        }
    }
    GraphToVisualize = g;
}

//Обробка події натискання кнопки відкритя нового файлу
private void Button_Click(object sender, RoutedEventArgs e)
{
    // Відкриття діалогового вікна для вибору файлу онтології
    OpenFileDialog openFileDialog = new OpenFileDialog();
    if (openFileDialog.ShowDialog() == true)
    {
        Owlfile o = new Owlfile(File.ReadAllText(openFileDialog.FileName), openFileDialog.SafeFileName);
        OwlList.owllist.Add(o);
        OwlList.NewCurrentOwl(o);


        DeviceList.connectionDeviceList.Clear();
        DeviceList.complist.Clear();
        DeviceList.allList.Clear();

        INode type = o.g.GetUriNode("rdf:type");

        INode m = o.g.GetUriNode("owl:Class");
        INode endDev = o.g.GetUriNode(":EndDevices");
        INode conDev = o.g.GetUriNode(":ConnectionDevices");
        INode joined = o.g.GetUriNode(":joined_with");

        //створення списку елементів комп'ютерної мережі
        foreach (Triple s in o.g.GetTriplesWithPredicateObject(type, endDev))
        {
            DeviceList.complist.Add(s.Subject);
```

```
            DeviceList.allList.Add(s.Subject);
        }


        foreach (Triple s in o.g.GetTriplesWithPredicateObject(type, conDev))
        {
            DeviceList.connectionDeviceList.Add(s.Subject);
            DeviceList.allList.Add(s.Subject);
        }

        foreach (INode node in DeviceList.allList)

            txtEditor.Text += ViewModel.SplitFunction(node.ToString());

        CreateGraphToVisualize(DeviceList.getAllList());
    }

}

// елемент керування що забезпечує перемикання між різніми збереженними файлами
private void ComboBox_Selected(object sender, SelectionChangedEventArgs e)
{
    ComboBox comboBox1 = (ComboBox)sender;
    Owlfile selectedItem = (Owlfile)comboBox1.SelectedItem;
    OwlList.NewCurrentOwl(selectedItem);

    DeviceList.connectionDeviceList.Clear();
    DeviceList.complist.Clear();
    DeviceList.allList.Clear();

    INode type = OwlList.currentOwl.g.GetUriNode("rdf:type");

    INode m = OwlList.currentOwl.g.GetUriNode("owl:Class");
    INode endDev = OwlList.currentOwl.g.GetUriNode(":EndDevices");
    INode conDev = OwlList.currentOwl.g.GetUriNode(":ConnectionDevices");
    INode joined = OwlList.currentOwl.g.GetUriNode(":joined_with");

    foreach (Triple s in OwlList.currentOwl.g.GetTriplesWithPredicateObject(type, endDev))
    {
        DeviceList.complist.Add(s.Subject);
        DeviceList.allList.Add(s.Subject);
    }

    foreach (Triple s in OwlList.currentOwl.g.GetTriplesWithPredicateObject(type, conDev))
    {
        DeviceList.connectionDeviceList.Add(s.Subject);
        DeviceList.allList.Add(s.Subject);
    }

    foreach (INode node in DeviceList.allList)

        txtEditor.Text += ViewModel.SplitFunction(node.ToString());

    CreateGraphToVisualize(DeviceList.getAllList());
}

//вивод на екран хаарктеристик обраного у списку елементу
private void ListBox1_Selected(object sender, SelectionChangedEventArgs e)
{
    ListBox listfox1 = (ListBox)sender;
    INode selectedItem = (INode)listfox1.SelectedItem;
    if (selectedItem != null)
    {
```

```
            ViewModel.DataProperties(OwlList.currentOwl.g, ViewModel.SplitFunction(selectedItem.ToString()),
txtEditor);
        }
    }

    private void ListBox2_Selected(object sender, SelectionChangedEventArgs e)
    {
        ListBox listBox2 = (ListBox)sender;
        INode selectedItem = (INode)listBox2.SelectedItem;
        if (selectedItem != null)
            {
                ViewModel.DataProperties(OwlList.currentOwl.g, ViewModel.SplitFunction(selectedItem.ToString()),
txtEditor);
            }
    }

    //створення порожнього файлу онтології
    private void FileCreation(object sender, RoutedEventArgs e)
    {
        Owlfile o = new Owlfile("d");
        o.g.CreateUriNode(new Uri("http://example.org/select#dfd"));
        OwlList.owllist.Add(o);

    }


    //додання нового елементу комп мережі
    private void Button_Add(object sender, RoutedEventArgs e)
    {

        DeviceList.connectionDeviceList.Clear();
        DeviceList.complist.Clear();
        DeviceList.allList.Clear();
        //зчитування інформації введеної з клавіатури
        ViewModel.AddComp(OwlList.currentOwl.g, DeviceText.Text, ViewModel.DeviceType);

        DeviceText.Text = ViewModel.DeviceType;

        INode type = OwlList.currentOwl.g.GetUriNode("rdf:type");
        INode c = OwlList.currentOwl.g.GetUriNode(":comp1");
        INode m = OwlList.currentOwl.g.GetUriNode("owl:Class");
        INode endDev = OwlList.currentOwl.g.GetUriNode(":EndDevices");
        INode conDev = OwlList.currentOwl.g.GetUriNode(":ConnectionDevices");
        INode joined = OwlList.currentOwl.g.GetUriNode(":joined_with");
        // створення нових семантичних триплетів
        // що містять опис нового пристрою
        foreach (Triple s in OwlList.currentOwl.g.GetTriplesWithPredicateObject(type, endDev))
        {
            DeviceList.complist.Add(s.Subject);
            DeviceList.allList.Add(s.Subject);
        }

        foreach (Triple s in OwlList.currentOwl.g.GetTriplesWithPredicateObject(type, conDev))
        {
            DeviceList.connectionDeviceList.Add(s.Subject);
            DeviceList.allList.Add(s.Subject);
        }

        foreach (INode node in DeviceList.allList)

            txtEditor.Text += ViewModel.SplitFunction(node.ToString());
        CreateGraphToVisualize(DeviceList.getAllList());
    }
```

```
//вибір типу пристрою що хоче добавити користувач
private void Device_Type_Selected(object sender, SelectionChangedEventArgs e)
{
   ComboBox DeviceType = (ComboBox)sender;
   TextBlock selectedItem = (TextBlock)DeviceType.SelectedItem;
   ViewModel.DeviceType = selectedItem.Text;
}

private void TextBox_TextChanged(object sender, TextChangedEventArgs e)
{

}

private void Button_Remove(object sender, RoutedEventArgs e)
{
   DeviceList.connectionDeviceList.Clear();
   DeviceList.complist.Clear();
   DeviceList.allList.Clear();

   INode type = OwlList.currentOwl.g.GetUriNode("rdf:type");
   INode c = OwlList.currentOwl.g.GetUriNode(":" + RemoveDeviceName.Text);
   INode m = OwlList.currentOwl.g.GetUriNode("owl:Class");
   INode endDev = OwlList.currentOwl.g.GetUriNode(":EndDevices");
   INode conDev = OwlList.currentOwl.g.GetUriNode(":ConnectionDevices");
   INode joined = OwlList.currentOwl.g.GetUriNode(":joined_with");

   IGraph toRemove = new Graph();
   //видалення пристрою та всіх зв'язків
   foreach (Triple s in OwlList.currentOwl.g.GetTriplesWithSubject(c))
   {

      toRemove.Assert(s);
   }

   foreach (Triple s in toRemove.Triples)
   {

      OwlList.currentOwl.g.Retract(s);
   }

   foreach (Triple s in OwlList.currentOwl.g.GetTriplesWithPredicateObject(type, endDev))
   {
      DeviceList.complist.Add(s.Subject);
      DeviceList.allList.Add(s.Subject);
   }

   foreach (Triple s in OwlList.currentOwl.g.GetTriplesWithPredicateObject(type, conDev))
   {
      DeviceList.connectionDeviceList.Add(s.Subject);
      DeviceList.allList.Add(s.Subject);
   }

   foreach (INode node in DeviceList.allList)

      txtEditor.Text += ViewModel.SplitFunction(node.ToString());

   CreateGraphToVisualize(DeviceList.getAllList());

}

//виклик вікна для вводу назви файлу
//який потрібно зберегти
private void SaveFile(object sender, RoutedEventArgs e)
```

```
        {

            Window1 enter = new Window1();
            enter.Show();
        }

        //додання та видалення зв'язків між пристроями мережі
        private void Button_Click_1(object sender, RoutedEventArgs e)
        {
            DeviceList.connectionDeviceList.Clear();
            DeviceList.complist.Clear();
            DeviceList.allList.Clear();

            ViewModel.AddConnection(OwlList.currentOwl.g, dev1.Text, dev2.Text);

            INode type = OwlList.currentOwl.g.GetUriNode("rdf:type");
            INode c = OwlList.currentOwl.g.GetUriNode(":comp1");
            INode m = OwlList.currentOwl.g.GetUriNode("owl:Class");
            INode endDev = OwlList.currentOwl.g.GetUriNode(":EndDevices");
            INode conDev = OwlList.currentOwl.g.GetUriNode(":ConnectionDevices");
            INode joined = OwlList.currentOwl.g.GetUriNode(":joined_with");

            foreach (Triple s in OwlList.currentOwl.g.GetTriplesWithPredicateObject(type, endDev))
            {
                DeviceList.complist.Add(s.Subject);
                DeviceList.allList.Add(s.Subject);
            }
            foreach (Triple s in OwlList.currentOwl.g.GetTriplesWithPredicateObject(type, conDev))
            {
                DeviceList.connectionDeviceList.Add(s.Subject);
                DeviceList.allList.Add(s.Subject);
            }

            foreach (INode node in DeviceList.allList)

                txtEditor.Text += ViewModel.SplitFunction(node.ToString());

            CreateGraphToVisualize(DeviceList.getAllList());
        }

        private void Button_Click_2(object sender, RoutedEventArgs e)
        {

        }

    }
}
```

```
<Window x:Class="OntoNet.MainWindow"
      xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
      xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
      xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
      xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
      xmlns:local="clr-namespace:OntoNet"
      xmlns:graphsharp="clr-namespace:GraphSharp.Controls;assembly=GraphSharp.Controls"
      xmlns:zoom="clr-namespace:WPFExtensions.Controls;assembly=WPFExtensions"
      xmlns:models="clr-namespace:OntoNet.Model"
      mc:Ignorable="d"
       x:Name="root"
      Title="OntoNet" Height="984.5" Width="1436" >
   <Window.Resources>
      <DataTemplate DataType="(x:Type models:OwlList)"></DataTemplate>
      <DataTemplate DataType="(x:Type models:DeviceList)"></DataTemplate>
   </Window.Resources>
   <Grid Margin="5,4,45,7">
      <Grid.ColumnDefinitions>
         <ColumnDefinition Width="342*"/>
         <ColumnDefinition Width="347*"/>
      </Grid.ColumnDefinitions>


      <Menu Height="25" VerticalAlignment="Top" Grid.ColumnSpan="2">
         <MenuItem Header="File">
            <MenuItem Header="New Project" Click="FileCreation" ></MenuItem>
            <MenuItem Header="Open Project from PC" Click="Button_Click" >
            </MenuItem>
            <MenuItem Header="Select saved project" HorizontalAlignment="Left" Width="200" >
               <ComboBox x:Name="comboBox1" DataContext="comboBoxDataContext"
                     HorizontalAlignment="Left" Margin="0,0,0,0" VerticalAlignment="Top" Width="140" Height="27"
                     SelectionChanged="ComboBox_Selected" DisplayMemberPath="FilePath"
                     ItemsSource="{Binding Mode=OneWay }"/>
            </MenuItem>
            <Separator />
            <MenuItem Header="Exit" ></MenuItem>
         </MenuItem>
         <MenuItem Header="Save Model" Click="SaveFile"  ></MenuItem>
         <MenuItem Header="View" ></MenuItem>
      </Menu>

      <TextBox Name="txtEditor" Margin="324,70,70,722" Grid.Column="1"
                  Width="300" Height="150"
       HorizontalScrollBarVisibility="Visible"
       VerticalScrollBarVisibility="Visible"
       TextWrapping="Wrap"
            RenderTransformOrigin="0.762,0.519" />

      <ListBox x:Name="listBox1" SelectionChanged="ListBox1_Selected"
            DataContext="listDataContext"
            ItemsSource="{Binding Mode=OneWay }"
            HorizontalAlignment="Left" Height="137" Margin="62,301,0,0" VerticalAlignment="Top" Width="266"
RenderTransformOrigin="2.96,1.88" Grid.Column="1"/>
      <ListBox x:Name="listBox2"
            SelectionChanged="ListBox2_Selected"
            DataContext="listDataContext"
            ItemsSource="{Binding Mode=OneWay }"
            HorizontalAlignment="Left" Height="137" Margin="371,307,0,0" VerticalAlignment="Top" Width="253"
RenderTransformOrigin="2.58,1.06" Grid.Column="1"/>
      <Label Content="List of end-devices" HorizontalAlignment="Left" Margin="79,278,0,0" VerticalAlignment="Top"
```

```
Width="114" Grid.Column="1" Height="26"/>
    <Label Content="List of Intermediary devices " HorizontalAlignment="Left" Margin="371,276,0,0"
VerticalAlignment="Top" Grid.Column="1" Height="26" Width="158"/>
    <TabControl Margin="-4,92,57,0" RenderTransformOrigin="-0.136,0.493">
      <zoom:ZoomControl Margin="0,0,4,0" Height="828" VerticalAlignment="Bottom">
        <graphsharp:GraphLayout x:Name="GraphLayout"
                    Graph="{Binding ElementName=root,Path=GraphToVisualize}"
                    LayoutAlgorithmType="FR"
                    OverlapRemovalAlgorithmType="FSA"
                    HighlightAlgorithmType="Simple" Margin="0,-7" />
      </zoom:ZoomControl>
    </TabControl>
    <Button Content="Add device" Grid.Column="1" Click="Button_Add" HorizontalAlignment="Left"
Margin="425,499,0,0" VerticalAlignment="Top" Width="75" RenderTransformOrigin="3.327,-0.5" Height="24"/>
    <TextBox Name="DeviceText" Grid.Column="1" Height="23" Margin="130,500,0,0" TextWrapping="Wrap"
VerticalAlignment="Top" HorizontalAlignment="Left" Width="120"/>
    <ComboBox x:Name="DeviceType"
        SelectionChanged="Device_Type_Selected"
        Grid.Column="1" HorizontalAlignment="Left" Margin="283,499,0,0" VerticalAlignment="Top" Width="120"
Height="24">
      <TextBlock Text="PC"/>
      <TextBlock Text="Server"/>
      <TextBlock Text="Bridge"/>
      <TextBlock Text="Hub"/>
      <TextBlock Text="Router"/>
    </ComboBox>
    <Label Content="New device name" Grid.Column="1" HorizontalAlignment="Left" Margin="130,467,0,0"
VerticalAlignment="Top" Height="28" Width="120"/>
    <Label Content="Device type" Grid.Column="1" HorizontalAlignment="Left" Margin="283,467,0,0"
VerticalAlignment="Top" Width="120" Height="27"/>
    <Button Content="Remove Device" Click="Button_Remove" Grid.Column="1" HorizontalAlignment="Left"
Margin="405,689,0,0" VerticalAlignment="Top" Width="95" Height="20"/>

    <Border BorderBrush="Black" BorderThickness="1" Grid.Column="1" HorizontalAlignment="Left" Height="54"
Margin="107,671,0,0" VerticalAlignment="Top" Width="467"/>
    <TextBox Name ="RemoveDeviceName" Grid.Column="1" HorizontalAlignment="Left" Height="20"
Margin="148,689,0,0" TextWrapping="Wrap" Text="&#xD;&#xA;" VerticalAlignment="Top" Width="198"
TextChanged="TextBox_TextChanged"/>
    <TextBox Name="dev1"  Grid.Column="1" HorizontalAlignment="Left" Height="29" Margin="130,767,0,0"
TextWrapping="Wrap"
        Text=""
        VerticalAlignment="Top" Width="144"/>
    <TextBox Name="dev2" Grid.Column="1" HorizontalAlignment="Left" Height="29" Margin="316,767,0,0"
TextWrapping="Wrap"
        Text=""
        VerticalAlignment="Top" Width="139"/>
    <Label Content="and" Grid.Column="1" HorizontalAlignment="Left" Margin="279,765,0,0"
VerticalAlignment="Top" RenderTransformOrigin="0.133,1.538" Height="26" Width="30"/>
    <Label Content="Connection between" Grid.Column="1" HorizontalAlignment="Left" Margin="138,741,0,0"
VerticalAlignment="Top" Height="26" Width="119"/>
    <Button Content="Add"
        Grid.Column="1" HorizontalAlignment="Left" Height="25" Margin="475,751,0,0" VerticalAlignment="Top"
Width="95"
        Click="Button_Click_1"/>
    <Button Content="Remove"
        Grid.Column="1" HorizontalAlignment="Left" Margin="475,781,0,0" VerticalAlignment="Top" Width="95"
Height="22"
        Click="Button_Click_2" RenderTransformOrigin="1.068,0.955"/>
    <Border BorderBrush="Black" BorderThickness="1" HorizontalAlignment="Left" Height="75"
Margin="107,741,0,0" VerticalAlignment="Top" Width="467" Grid.Column="1"/>
    <Label Content="IP" Grid.Column="1" HorizontalAlignment="Left" Margin="130,525,0,0"
VerticalAlignment="Top"/>
    <TextBox HorizontalAlignment="Left" Height="23" Margin="130,548,0,0" TextWrapping="Wrap"
```

```
VerticalAlignment="Top" Width="120" Grid.Column="1"/>
        <TextBox Grid.Column="1" HorizontalAlignment="Left" Height="23" Margin="283,548,0,0"
TextWrapping="Wrap" VerticalAlignment="Top" Width="120"/>
        <TextBox Grid.Column="1" HorizontalAlignment="Left" Height="23" Margin="425,548,0,0"
TextWrapping="Wrap" VerticalAlignment="Top" Width="120"/>
        <TextBox HorizontalAlignment="Left" Height="23" Margin="130,594,0,0" TextWrapping="Wrap"
VerticalAlignment="Top" Width="120" Grid.Column="1"/>
        <TextBox Grid.Column="1" HorizontalAlignment="Left" Height="23" Margin="283,594,0,0"
TextWrapping="Wrap" VerticalAlignment="Top" Width="120"/>
        <TextBox Grid.Column="1" HorizontalAlignment="Left" Height="23" Margin="425,594,0,0"
TextWrapping="Wrap" VerticalAlignment="Top" Width="120"/>
        <TextBox Grid.Column="1" HorizontalAlignment="Left" Height="23" Margin="30,107,0,0" TextWrapping="Wrap"
VerticalAlignment="Top" Width="120"/>
        <TextBox Grid.Column="1" HorizontalAlignment="Left" Height="23" Margin="172,107,0,0"
TextWrapping="Wrap" VerticalAlignment="Top" Width="120"/>
        <TextBox Grid.Column="1" HorizontalAlignment="Left" Height="23" Margin="30,176,0,0" TextWrapping="Wrap"
VerticalAlignment="Top" Width="120" RenderTransformOrigin="0.15,0.609"/>
        <Button Content="Remove" Grid.Column="1" HorizontalAlignment="Left" Margin="172,177,0,0"
VerticalAlignment="Top" Width="102"/>
        <Button Content="Create" Grid.Column="1" HorizontalAlignment="Left" Margin="197,135,0,0"
VerticalAlignment="Top" Width="51"/>
        <Label Content="Add ontology class" Grid.Column="1" HorizontalAlignment="Left" Margin="29,74,0,0"
VerticalAlignment="Top"/>
        <Label Content="Subclass of" Grid.Column="1" HorizontalAlignment="Left" Margin="189,74,0,0"
VerticalAlignment="Top"/>
        <Label Content="Remove ontology class" Grid.Column="1" HorizontalAlignment="Left" Margin="27,152,0,0"
VerticalAlignment="Top"/>
        <Label Content="Connection_Length" Grid.Column="1" HorizontalAlignment="Left" Margin="290,526,0,0"
VerticalAlignment="Top"/>
        <Label Content="CableType" Grid.Column="1" HorizontalAlignment="Left" Margin="429,526,0,0"
VerticalAlignment="Top"/>
        <Label Content="OperatingSystem" HorizontalAlignment="Left" Margin="130,568,0,0" VerticalAlignment="Top"
Grid.Column="1"/>
        <Label Content="SSID" Grid.Column="1" HorizontalAlignment="Left" Margin="428,568,0,0"
VerticalAlignment="Top"/>
        <Label Content="MAC Adress" Grid.Column="1" HorizontalAlignment="Left" Margin="291,571,0,0"
VerticalAlignment="Top"/>
        <Border BorderBrush="Black" BorderThickness="1" Grid.Column="1" HorizontalAlignment="Left" Height="171"
Margin="107,467,0,0" VerticalAlignment="Top" Width="467"/>
    </Grid>
</Window>
```

ДОДАТОК 3

Створення тематичного реєстру та візуалізація комп’ютерних мереж

Опис програмного модулю

УКР.НТУУ”КПІ ім. Ігоря Сікорського”_ТЕФ_АПЕПС_ТВ6151_20Б 13-1

Аркушів 8

Київ – 2020

АНОТАЦІЯ

Додаток надає інформацію про структуру та функції програмного модулю обробки тематичного реєстру та візуалізації комп’ютерних мереж.

Розроблене програмне забезпечення дозволяє взаємодійяти з поняттями онтології, не володіючи специфічними знаннями та перетворювати дані про онтологію з внутрішнього представлення системи у візуалізацію у формі графа .

Модуль було створено з використанням інтегрованого середовища розробки Microsoft Visual Studio, бібліотек та мови програмування C#.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

Програмний модуль має назву OntoNet, утворено від Ontology та Network. Застосунок працює незалежно від мережі інтернет, працює з онтологічними моделями що знаходяться на комп’ютері користувача.

Програма була написана мовою програмування C# та мовою розмітки XAML. При цьому використовувалися бібліотеки dotNetRdf, QuickGraph, GraphSharp.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

Створене програмне рішення формує зображення онтологічного графу відповідно до поточної моделі комп’ютерної мережі, та змінює це зображення, якщо користувач власноруч змінив онтологію за допомогою елементів інтерфейсу програми.

Окремі компоненти модулю відповідають за реалізацію наступних функцій:

— запису інформації про ребра та вершини онтологічного графу до проміжного подання у вигляді списку онтологічних триплетів;

— запису інформації з графу триплетів до графу візуалізації;

— змінення поточної онтологічної моделі;

— перезапис зміненної моделі до файлу;

— вибір будь-якого файлу, що був завантажений до програми.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Структура програми складається з клієнтського інтерфейсу та програмного модулю, який можна розділити на менші модулі за призначенням.

Загальна логіка роботи додатку виглядає наступним чином:

1) Завантаження або створення онтологічної моделі;
2) Візуалізація комп’ютерної мережі у вигляді графу;
3) Виведення на екран інформації про елементи мережі;
4) Редагування моделі, видалення та додавання пристроїв, зв’язків між ними та негайне відображення змін на графі
5) Збереження отриманої моделі для подальшого використання цією або сторонніми програмами.

ТЕХНІЧНІ ЗАСОБИ, ЩО ВИКОРИСТОВУВАЛИСЯ

Для створення системи були використані засоби мови програмування C# , фреймворку .NET WPF та бібліотек dotNetRdf, QuickGraph, GraphSharp.

Задачі дизайну інтерфейсу користувача були вирішені за допомогою мови розмітки XAML, за допомогою якої описані елементи керування WPF.

ВХІДНІ І ВИХІДНІ ДАНІ

Вхідними даними є:

—онтологічна модель реєстру у форматі OWL

Вихідними даними є:

— графічне зображення комп’ютерної мерпежі у вигляді графу.

— редагована онтологічна модель у форматі OWL