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

Кафедра автоматизації проектування енергетичних процесів і систем

До захисту допущено:

Завідувач кафедри

_____ Олександр Коваль

«___»_____2020 р.

Дипломна робота

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Геометричне моделювання в інформаційних системах»

спеціальності 122 «Комп'ютерні науки та інформаційні технології»

на тему: «Онтологічна система інформаційних ресурсів кафедри»

Виконав:
студент IV курсу, групи ТР-61
Круглий Дмитро Владиславович _____

Керівник:
старший викладач
Дацюк Оксана Антонівна _____

Консультант

_____

Рецензент:

к.т.н. доцент Сірий О.А. _____

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

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 122 Комп’ютерні науки та інформаційні технології

Спеціалізація Геометричне моделювання в інформаційних системах

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

ЗАВДАННЯ
на дипломну роботу студенту

Круглого Дмитра Владиславовича
(прізвище, ім’я, по батькові)

1. Тема роботи «Онтологічна система інформаційних ресурсів кафедри»

керівник роботи Дацюк Оксана Антонівна, старший викладач
(прізвище, ім’я, по батькові науковий ступінь, вчене звання)

затверджена наказом вищого навчального закладу від ”25” травня 2020 р. № **1268-с**

2. Строк подання студентом роботи 15.06.2020

3. Вихідні дані до роботи програмний продукт було розроблено в середовищі Visual Studio 2017 мовою програмування C#, з використанням платформи .NET Framework

4.Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) Створити дві онтології предметних областей: навчальні дисципліни та методичні матеріали, створити SPARQL запити до створених попередньо онтологій. Розробити програмний продукт, який дасть змогу користувачу знаходити швидко і легко необхідні дані в онтологіях шляхом відправки SPARQL запитів до даних онтологій.

5. Перелік ілюстративного матеріалу
«Мета роботи», «Актуальність», «Поставлена задача», «Огляд предметної області», «Створення онтологій», «Заповнення онтологій даними», «SPARQL запит», «Взаємодія компонентів системи», «Вибір програмного забезпечення», «Алгоритм роботи програми», «Створення інтерфейсу користувача», «Приклад роботи програми», «Висновки».

6. Дата видачі завдання ”14” жовтня 2020 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломної роботи	Термін виконання етапів роботи	Примітки
1.	Затвердження теми роботи	11.10.2019	
2.	Вивчення та аналіз задачі	14.10.2019 – 27.12.2019	
3.	Розробка архітектури та загальної структури системи	03.02.2020 – 06.03.2020	
4.	Розробка структур окремих підсистем	09.03.2020 – 10.04.2020	
5.	Програмна реалізація системи	13.04.2020 – 15.05.2020	
6.	Оформлення пояснювальної записки	18.05.2020 – 05.06.2020	
7.	Захист програмного продукту	09.06.2020	
8.	Передзахист	09.06.2020	
9.	Захист	15.06.2020	

Студент _____ Круглий Д.В.
(підпис) (прізвище та ініціали,)

Керівник роботи _____ Дацюк О.А.
(підпис) (прізвище та ініціали,)

АНОТАЦІЯ

Метою дипломної роботи є створення системи доступу до інформаційних ресурсів з використанням онтологічного підходу на основі предметної області та розробка програмного додатку для взаємодії з даними онтологіями.

Об’єктом дослідження для даних онтологій є навчальні дисципліни та методичні матеріали кафедри АПЕПС. В ході виконання роботи розглянуто існуючі системи для створення онтологій та виявлено переваги та недоліки кожної. Було створено програмний продукт для взаємодії з онтологіями та відправки SPARQL запитів до кожної з онтологій. А також було подано тези на конференцію «Сучасні проблеми наукового забезпечення енергетики».

Загальний обсяг даної роботи: 60 сторінок, 43 ілюстрації, 23 бібліографічні найменування.

Ключові слова: онтологія, SPARL, Visual Studio, Protégé, .Net Framework, owl.

ABSTRACT

The purpose of the thesis is to create a system of access to information resources using an ontological approach based on the subject area and the development of a software application for interaction with these ontologies.

The object of research for these ontologies are academic disciplines and methodological materials of the Department of APEPS. In the course of the work the existing systems for creating ontologies are considered and the advantages and disadvantages of each are revealed. A software product was created to interact with ontologies and send SPARQL queries to each of the ontologies. Theses were also submitted to the conference "Modern problems of scientific support of energy".

The total volume of this work: 60 pages, 43 illustrations, 23 bibliographic titles.

Keywords: ontology, SPARL, Visual Studio, Protégé, .Net Framework, owl.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ, УМОВНИХ ПОЗНАЧЕНЬ І ТЕРМІНІВ

1. OWL – мова опису онтологій для семантичної павутини. Дозволяє описувати класи та відношення між ними.
2. Онтологія – формалізоване представлення знань про певну предметну область.
3. C# – об’єктно орієнтована мова програмування.
4. Visual Studio – інтегроване середовище розробки програмного забезпечення.
5. Protégé – вільний, відкритий редактор онтологій і фреймворк для побудови бази знань.
6. SPARQL – мова формування запитів до баз даних знань.
7. .NET Framework – платформа для розробки програмного забезпечення мовою програмування C#.
8. RDF – технологія семантичної павутини, яка включає в себе середовище опису ресурсів, визначає загальну архітектуру метаданих і призначена для забезпечення сумісності метаданих за допомогою спільної семантики.

ВСТУП

З року в рік під час роботи певного підприємства збільшується накопичення обсягу даних на цьому підприємстві. На перший етапах це не є проблемою, адже отриманий розмір інформації є не настільки великим, щоб сповільнювати роботу системи. Але з часом інформація накопичується і виникають питання автоматизувати обробку даних, які затримують робочий процес. З одного боку, питання автоматизації робочого процесу на даний час для всіх не є новим, але з іншого, якщо розглянути його більш детально, то виникають складнощі. Тому використання онтологій в даній роботі зумовлено тим, що ієрархічна база даних, яка міститься в онтологічному файлі, є більш гнучкою. Тож розглянемо детальніше це питання.

Під «онтологією» ми розуміємо формалізоване представлення знань про певну предметну область. Вона включає в себе набір тверджень, понять, маючи інформацію про які, ми можемо побудувати класи, об’єкти до цих класів, функції, а також відношення. Найбільш широковживаною мовою запитів до онтологій є SPARQL. Ця мова дозволяє вибирати дані з системи за необхідним критерієм, який визначає користувач. Крім того, синтаксис цієї мови такий, що зразу ж дозволяє визначити, до якого класу, об’єкту, функції потрібно звертатися в системі навіть ще до запуску програмного застосунку.

Для виконання даної роботи бралася ідея, що було б дуже зручно мати програму, за допомогою якої можна швидко знайти інформаційні ресурси, пов’язані з навчальним процесом, переглянути документи (питання до екзамену, навчальні посібники) викладачів, які викладають на кафедрі. Це дозволить зекономити час та ресурси, адже шукати певний документ чи «скільки було екзаменів у групи ТР-61 в минулому семестрі?» на просторах інтернету займає певний час. Також дуже великий інтерес викликав «нестандартний» підхід до зберігання даних: замість звичайної бази даних

використати онтологію та SPARQL запити, працювати з якими, раніше не було просто ніякої можливості.

Основними завданнями, які були поставлені перед початком роботи є:

- ознайомлення із відкритим редактором онтологій Protégé 5.5.0;
- аналіз навчального процесу кафедри;
- розбиття навчального процесу кафедри на дві онтологічні структури;
- побудова дерева класів та object і data properties для кожної онтології;
- структурувати дані в онтологіях;
- написати програму для обробки SPARQL запитів до створених онтологій.

1. ПОСТАНОВКА ЗАДАЧІ

За всі свої часи існування, кафедра зазнала суттєвих змін. Якщо раніше майже вся інформація зберігалася в паперовому вигляді, то з часом вона почала переноситись на комп'ютери та іншу електронну техніку. Таким чином з'явилася потреба в швидкій обробці та швидкому пошуку інформації, що в свою чергу говорить нам про те, що також необхідний інтерфейс, за допомогою якого можна було б здійснювати пошук та аналіз інформації. Також не потрібно забувати про те, що якщо на етапі створення база даних не була добре продумана, не була розрахована завантаженість цієї бази з певним часом, то скоро шукати в таблицях бази інформацію буде дуже і дуже довго, адже кожного року кількість інформації в цій базі збільшується.

Тому основною задачею даної роботи є організація пошуку інформації в таких базах та внесення періодично нових даних до сховища. Більше того, акцент було зроблено на створенні простого інтерфейсу для звичайного користувача, навіть такого, який взагалі не знайомий з програмуванням та базами даних, що дозволило б йому знайти потрібну йому інформацію за досить короткий проміжок часу.

Вся інформація про навчальні матеріали та документи буде зберігатися в файлах формату .owl. Також пропонується створити 2 файли такого формату, кожен з яких буде відповідати за свою предметну область. Це дозволить розробнику краще орієнтуватися під час написання та відправлення SPARQL запитів. Користувач же в такому випадку отримає систему, в якій зможе дуже швидко знайти необхідне, адже дані будуть відсортовані та належатимуть своїй предметній області, що не спричинятиме перевантаження бази.

Дану задачу можна розбити на такі підзадачі:

- аналіз предметної області;
- створення структури онтології навчальних дисциплін та онтології документів;

- побудова спроектованих онтологій;
- написання SPARQL запитів до онтологій;
- створення інтерфейсу для користувача;
- відображення знайденої інформації користувачеві.

Однією із найголовніших задач ще до створення програмного продукту, є визначення того, як саме він буде експлуатуватися. Розроблений в цій роботі додаток дасть змогу користувачу:

- робити запити до онтології;
- отримувати запитувані дані;
- відкривати знайдені документи прямо з вікна програми;
- переглядати інформацію про навчальний процес студентів;
- робити аналіз за знайденими матеріалами.

Однією із основних вимог до майбутнього продукту є можливість відправки будь-яким користувачем SPARQL запитів до онтології. Це дає зможу всім користувачами, які незнайомі з онтологіями та програмування, легко знаходити потрібну інформацію.

1.1. Висновки до розділу

Таким чином, була поставлена задача розробки онтологічної інформаційної системи для підтримки навчального процесу кафедри. В результаті впровадження, дану програму зможуть використовувати як і викладачі (для перегляду навчального завантаження студентів), так і студенти (для пошуку необхідних матеріалів та документів по певному предмету).

2. АНАЛІЗ ЗАДАЧІ ПРЕДСТАВЛЕННЯ ОНТОЛОГІЙ ІНФОРМАЦІЙНОЇ СИСТЕМИ

Відповідно до інформації, наведеної у попередньому розділі, виникає потреба створення структури майбутньої інформаційної системи навчальних ресурсів кафедри у вигляді онтологій. У подальшому це значно спростить роботу при побудові запитів клієнтом для отримання даних.

2.1. Візуалізація предметної області

Для того, щоб зрозуміти на основі чого будувати онтології, необхідно спочатку розібратися з предметною областю. Для побудови майбутніх онтологій з даними було вирішено за основу взяти методичні матеріали та наукові ресурси, а також навчальні плани впродовж всіх семестрів кафедри.

2.1.1. Опис методичних матеріалів кафедри

Для початку, необхідно розібратися з тим, що ж собою представляють методичні матеріали та інформаційні ресурси, адже при побудові онтології за певною предметною областю, перш за все, потрібна актуальність і відповідність даних.

Методичними матеріалами називають матеріали, або ж документи, в яких містяться вказівки та пояснення, виконання яких, на думку викладача, має сприяти найбільш ефективному вивченню та засвоєнню навчальної програми. В цілому, їх можна розділити на 4 групи: інформаційні, описуючі, інструктуючі та прикладні. Першу групу можна описати досить коротко. Їх головна задача – донести інформацію до адресата. Ці методичні матеріали не ставлять за мету пояснити прийоми і методи, проаналізувати чийсь досвід, описати дії. Як правило, в інформаційних матеріалах переважає

перелічувальна інформація, цифровий звіт. Завдання другої групи методичних матеріалів – це висвітлити досвід, розповісти про проведену справу. Не завжди такого роду матеріали існують самостійно, частіше – у складі рекомендацій, розробок. Призначенням інструктуючих матеріалів є роз'яснення мети і порядку дії, методики організації, проведення справи, акції, показати можливі прийоми, форми. Остання група ж має безпосередньо практичне значення, лише умовно називається прикладною [1].

Інформаційними ресурсами в свою ж чергу називають дані в будь-якому вигляді, що зазвичай багаторазово використовуються для вирішення певного роду проблем. Тобто, це може бути як книга, документ, файл, фото, сайт, відео. Для інформаційних ресурсів в інтернеті характерний певний час життя і доступність більш, ніж одному користувачеві. Також інформаційним ресурсом можна назвати окремо взятий сайт, портал або кілька інтернет-проектів. До них також можна віднести окремі документи, або окремі масиви документів в інформаційних системах [2]. Але так, як завдання полягає в побудові онтології для методичних матеріалів та інформаційних ресурсів кафедри, то необхідно розібратися ще в тому, яке відношення вони мають безпосередньо для кафедри. Інформаційними ресурсами кафедри зазвичай є вся інформація, якою володіє кафедра, яка є необхідною для організації навчального процесу. До неї можна віднести: навчальні плани (про які далі буде йти мова), робочі навчальні програми, програми до навчальних дисциплін, підручники, конспекти, навчальні посібники, методичні та навчальні посібники для курсових робіт, екзаменів, лабораторних та самостійних робіт, план проведення занять, лабораторних, консультацій, теми для дипломних проектів та курсових робіт, завдання для проведення семестрового контролю та поточного контролю, критерії оцінювання дипломних, контрольних, курсових та самостійних робіт, матеріали для дистанційного навчання. Обсяг наведеної вище інформації дуже і дуже великий. Її обробка займає досить багато часу, а якщо ще і врахувати той факт, що інформації з кожним роком стає все більше й більше, то виникає потреба у створенні продуманої бази, яка буде зберігати дані про інформаційні ресурси

кафедри та не буде затримувати роботу викладачів/студентів під час обробки цих даних. Тому найбільш оптимальним і зручним варіантом у цьому випадку буде використання онтології.

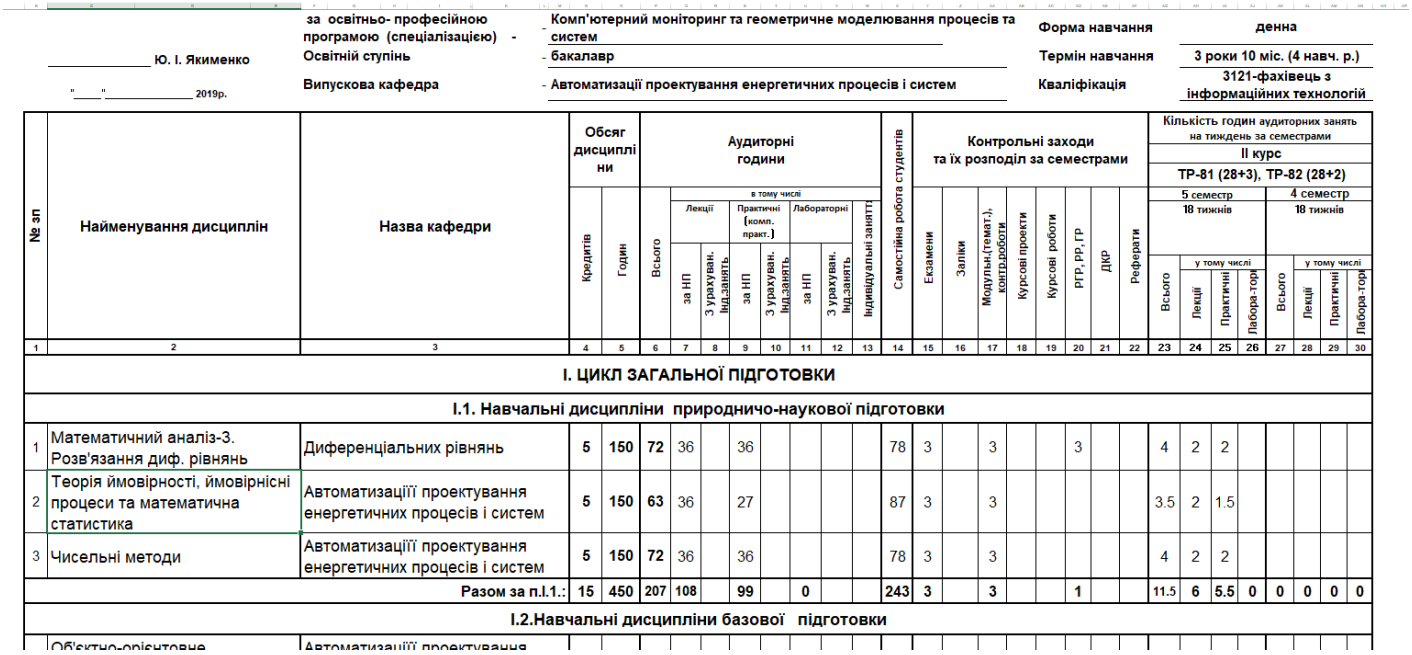
2.1.2. Опис навчальних планів кафедри

Тепер що ж до навчальних планів кафедри. Це документ, що встановлює список предметів, обсяг робочого часу, порядок проходження і окремі етапи підготовки навчальних дисциплін, стажувань, факультативних занять, навчальних модулів і інших видів практичної освітньої діяльності. Також в ньому вказуються положення про проміжні оцінки студентів, учнів. Кожен навчальний заклад готує навчальний план і програму навчання самостійно. Їх схвалення і впровадження відповідає тому порядку, який зафіксований у статуті навчального закладу. Викладачі можуть проявляти творчу ініціативу в складанні плану, створювати і використовувати власні авторські програми і методи навчання в межах окремо взятої навчальної програми [3].

Будь-який навчальний план впроваджується на підставі освітньо-професійної програми та визначає:

- вид заняття та його обсяг;
- перелік та обсяг постановлених дисциплін та дисциплін за вибором студентів, а також порядок їх вивчення;
- обсяг загально часу на вивчення предмету протягом семестру, часу на лабораторні та практичні заняття, а також самостійну роботу студента;
- термін та вигляд проведення навчальної практики;
- проведення атестацій;
- графік навчального процесу;
- проведення семестрового контролю [4].

Навчальні плани кафедри АПЕПС зберігаються у файлі, створеному в середовищі Excel (рисунок 2.1).



_____ Ю. І. Якименко

"____" _____ 2019р.

за освітньо- професійною програмою (спеціалізацією) - Комп'ютерний моніторинг та геометричне моделювання процесів та систем

Освітній ступінь - бакалавр

Випускова кафедра - Автоматизації проектування енергетичних процесів і систем

Форма навчання: денна

Термін навчання: 3 роки 10 міс. (4 навч. р.)

Кваліфікація: 3121-фахівець з інформаційних технологій

№ зп	Найменування дисциплін	Назва кафедри	Обсяг дисципліни: Кредитів	Обсяг дисципліни: Годин	Аудиторні години: Всього	в тому числі: Лекції: за НП	Лекції: З урахуван. Інд.занять	Практичні (комп., практ.): за НП	Практичні (комп., практ.): З урахуван. Інд.занять	Лабораторні: за НП	Лабораторні: З урахуван. Інд.занять	Індивідуальні заняття	Самостійна робота студентів	Контрольні заходи та їх розподіл за семестрами: Екзамени	Заліки	Модульн.(темат.), контр.роботи	Курсові проекти	Курсові роботи	РГР, РР, ГР	ДКР	Реферати	Кількість годин аудиторних занять на тиждень за семестрами: II курс ТР-81 (28+3), ТР-82 (28+2) 5 семестр 18 тижнів: Всього	у тому числі: Лекції	Практичні	Лабора-тор	4 семестр 18 тижнів: Всього	у тому числі: Лекції	Практичні	Лабора-тор
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
І. ЦИКЛ ЗАГАЛЬНОЇ ПІДГОТОВКИ																													
І.1. Навчальні дисципліни природничо-наукової підготовки																													
1	Математичний аналіз-3. Розв'язання диф. рівнянь	Диференціальних рівнянь	5	150	72	36		36					78	3		3			3			4	2	2					
2	Теорія ймовірності, ймовірнісні процеси та математична статистика	Автоматизації проектування енергетичних процесів і систем	5	150	63	36		27					87	3		3						3.5	2	1.5					
3	Чисельні методи	Автоматизації проектування енергетичних процесів і систем	5	150	72	36		36					78	3		3			1			4	2	2					
	Разом за п.І.1.:		**15**	**450**	**207**	**108**		**99**		**0**			**243**	**3**		**3**			**1**			**11.5**	**6**	**5.5**	**0**	**0**	**0**	**0**	**0**
І.2.Навчальні дисципліни базової підготовки																													
	Об'єктно-орієнтоване	Автоматизації проектування																											

Рисунок 2.1 – Навчальний план кафедри АПЕПС

Під час побудови навчальних планів, вся інформація про них зберігається в реляційних базах даних. Але з року в рік інформація оновлюється і обсяг даних, з якими необхідно працювати, дедалі стає все більшим та більшим, що значно ускладнює і затримує пошук необхідної інформації по базі. Тому в якості сховища для зберігання даних про навчальні плани була використана онтологія.

2.2. Поняття онтології

Онтологія – це явна конкретизація концептуалізації. Термін запозичено з філософії, де онтологія – це систематичний вклад в існування. В інформатиці ж онтологією являється формальне подання знань, сукупність понять та зв’язки між цими поняттями. Вона забезпечує спільну лексику, яка може використовуватися для моделювання даних та об’єктів, а також властивостей та відношень між ними. Комп'ютерна онтологія позначає інтерпретацію групи ідей у певній галузі, яка визначає взаємозв'язок між цими ідеями. Онтологія може бути використана для вивчення існування сутностей у межах певного домену, а іноді може бути використана для

ідентифікації самого домену. В контексті комп’ютерів онтологія виступає як структурна схема організації даних. Вона широко використовується для організації інформації та концепцій у таких сферах, як штучний інтелект, системи, семантика та біомедична та інформаційна архітектура. Значення певної інформації, як правило, виражається на основі концептуальної інформаційної моделі, яка використовується для моделювання додатків та структурування даних. Основні поняття, що використовуються для побудови таких моделей, включають сутність, діяльність, елемент та призначення. Концептуальні моделі визначають семантичні терміни та механізми організації інформації, роблячи набір припущень про фактичні програми, що підлягають моделюванню. Наприклад, якщо передбачається, що додаток включає взаємопов'язані сутності, концептуальна модель визначає такі терміни, як властивість та відношення. Наприклад, загальна мова алгебраїчних специфікацій – це фактично стандарт у специфікації програмного забезпечення, який також вважається мовою онтології. Він кодує специфікації для модульності та структурування програмного забезпечення з метою включення в безліч інших існуючих мов специфікацій [5].

Першим онтології почав досліджувати Томас Грубер [6]. В своїх перших онтологіях він розглядав аспекти для взаємодії з людиною та між інтелектуальними системами. Інтелектуальною системою ж називають програму, яка моделює певні аспекти для інтелектуальної діяльності особи. Звичайно, моделювання виконується будь-якою програмою в певній іншій мірі, адже в основному і полягає найбільша цінність комп’ютера для людини – комп’ютер дозволяє звільнити людину від виконання якоїсь однотипної діяльності. Ця діяльність може бути досить складною і витонченою, але вона завжди однотипна: комп’ютерна система, створена, наприклад, для редагування графіки, не може бути використана для управління комбайнами під час сінокосу. У цьому сенсі знання, які закладає в програму її творець (тобто алгоритм цієї програми), завжди статичні, вони не переінакшуються (лише, за винятком конкретної інформації, яка називається «дані програми»). Інтелектуальна система в цьому сенсі стає більш

універсальною - в неї закладено інформацію про те, що необхідно виконувати в процесі роботи програми, що не вшито в програму раз і назавжди, але має можливість мінятися. В такому випадку, цю інформацію необхідно передати до програми як дані, в результаті чого виникає необхідність для того, щоб їх описати.

В структуру звичайної онтології входять:

- аксіоми;
- відносини;
- екземпляри;
- поняття.

Аксіомами визначаються і записуються відношення, які завжди є правдивими. Серед основних аксіом, що визначають формальну семантику понять, можна виділити:

- аксіоми тотожності (equivalent-to). Формально вони визначають рівність множин інтерпретації класів.

- аксіоми успадкування (subclass / superclass). Визначають входження безлічі інтерпретації одного класу (підкласу) в безліч іншого (суперкласу).

- аксіома незв’язності (disjoint) класів, вказує на те, що безлічі інтерпретації класів не перетинаються.

- аксіома перерахування (enumeration), дозволяє задати клас екстенсіонально, тобто шляхом перерахування всіх його примірників [7].

Відношеннями пов’язують класи онтології, завдяки чому їх можна описувати. Належність до певної категорії є одним із найпоширеніших типом відношень.

Екземплярами ж називають конкретних представників певної категорії, представників певного класу, певних сутностей або ж явищ.

Поняттям же позначають опис та визначення чогось конкретного, якоїсь сутності. Його завжди описують класами онтології [8].

2.3. Засоби опису онтологій

Мова опису онтології – це формальна мова, яка використовується для кодування онтології. В цьому пункті буде розглянуто лише найбільш популярні на використовувані мови.

2.3.1. Мова опису OWL

OWL – мова для семантичної павутини. Веб-мова онтології (OWL) була розроблена (або, точніше, отримана з декількох більш ранніх мовних ініціатив), щоб забезпечити стандартизований спосіб представлення онтологій у семантичній мережі. Сама OWL постачається у трьох мовних варіантах: OWL Lite, OWL DL (опис логіки) та OWL Full (W3C 2009b). Ці варіанти виявляють зростаючий рівень обчислювальної складності та експресивності. Усі варіанти беруть початок у давній традиції дослідження представлення знань та машинного навчання, що завершилось описом логіки у 1980-х та 1990-х роках. RDF був розроблений для різних, більш прагматичних цілей обміну даними між системами, і на початку історії семантичної веб-мережі були зроблені зусилля для гармонізації RDF з OWL – оскільки RDF дозволяє широкий спектр конструкцій, він більш виразний, ніж OWL Full, але також обчислювально нерозбірливий. І OWL, і RDF можуть бути надані в різних синтаксисах, таких як XML, N-Triples і Notation3 (Berners-Lee 2009). Незважаючи на те, що всі ці синтаксиси легко читаються та записуються, на практиці файли RDF та OWL зазвичай створюються засобами редагування онтології, такими як Protégé (Gennari et al., 2003) [9].

Мета результуючої онтології OWL полягає у наданні на основі стандартів моделі деякого набору фактів про світ. Саме тому, що це стандарт, інші системи можуть обробляти онтологію OWL та робити висновки про неї; що важливо, не лише щодо самої конкретної онтології, але й про те, як ця онтологія могла би з'єднуватися з іншими. Семантична павутина передбачає мережу онтологій, які функціонують як велика,

розподілена база даних. Звичайно, можна сказати, що поточна павутина надає це в більш аморфній формі, але вся суть семантичної павутини полягає саме в тому, що будь-які факти в онтології мають певний контекст, який стає явним.

2.3.2. Фреймворк опису ресурсів RDF

RDF було розроблено для полегшення сумісності метаданих, особливо у царині всесвітньої павутини. Оскільки метадані охоплюють занадто велику різноманітність інформації для їх вичерпного та категоричного визначення, RDF слідує за лідерами XML, але не надає набір можливостей, він надає засоби для опису дійсної системи. RDF - це спосіб моделювання як ресурсу всього, що можна представити як Універсальний ідентифікатор ресурсу. RDF та мова запитів SPARQL були визнані двома ключовими технологіями Semantic Web. Репозиторій RDF - це сукупність трійки, позначених як <предмет, предикат, об’єкт> і можуть бути представлені у вигляді графа. Вершини цього графа позначають суб’єкти та об’єкти, а ребра позначають предикати. Хоча SPARQL є стандартним способом доступу до даних RDF, він залишається виснажливим і складним для непрофесійних користувачів через складність синтаксису SPARQL та схеми RDF [10]. Щоб автоматично генерувати запит SPARQL, система повинна була б розділити вхід користувача на синтаксичні маркери та лексеми, зіставити лексеми на поняття в онтології, зв’язати ідентифіковані поняття на основі зв’язків в онтології та задати запит для збору результатів.

2.3.3. Схема опису RDFS

Схема RDF (RDFS) пропонує конструкції для визначення класів, ієрархій класів та належності ресурсів у класах. Схема RDF також дозволяє оголосити обмеження на домени та діапазони властивостей. RDF-схема передує мові веб-онтології, яку ми коротко висвітлюємо, і вона отримала прийняття самостійно. Важливо зазначити, що

RDF не застосовує жодної з семантики своїх конструкцій. RDF - це просто мова, і це стосується впровадження систем RDF, щоб використовувати їх семантику.

Конструкції схеми RDF самі описуються словником RDF за адресою http://www.w3.org/2000/01/rdf-schema#, а його кваліфікована назва - rdfs:. Найбільш основна конструкція RDFS - це rdfs:type, яка використовується для позначення належності ресурсу до класу [11].

Підкласи в RDFS можна оголосити за допомогою конструкції rdfs:subclassOf, наприклад: ex:Comedy rdfs:subclassOf ex:Movie . Слід зауважити, що твердження датуються, як сполучники, але якщо додати: ex:Comedy rdfs:subclassOf ex:LaughProvokingEvents, тоді Comedy буде вважатися, що знаходить в перетині між Movie та LaughProvokingEvents [12].

2.3.4. Мова опису DAML+OIL

Мова DAML + OIL призначена для опису структури даних; вона використовує об'єктно-орієнтований підхід, описуючи структуру з точки зору класів та властивостей. Онтологія складається з набору аксіом, які стверджують, наприклад, відношення між класами або властивостями. Затверджуючи, що ресурси (пари ресурсів) є екземплярами класів (властивостей) DAML + OIL, залишається RDF, завдання для якого воно добре підходить. Коли ресурс r є екземпляром класу C, ми кажемо, що r має тип C. З формальної точки зору DAML + OIL може розглядатися як еквівалентний дуже виразній логіці опису (DL) з DAML + OIL онтологія, що відповідає термінології DL (Tbox). Як і в DL, класи DAML + OIL можуть бути іменами (URI) або виразами, і надаються різні конструктори для побудови виразів класу. Експресивна сила мови визначається підтримуваними конструкторами класу (рисунок 2.2) та типами, що підтримуються аксіомою [13].

Constructor	DL Syntax	Example
intersectionOf	$C_1 \sqcap \ldots \sqcap C_n$	Human ⊓ Male
unionOf	$C_1 \sqcup \ldots \sqcup C_n$	Doctor ⊔ Lawyer
complementOf	$\neg C$	¬Male
oneOf	$\{x_1 \ldots x_n\}$	{john, mary}
toClass	$\forall P.C$	∀hasChild.Doctor
hasClass	$\exists P.C$	∃hasChild.Lawyer
hasValue	$\exists P.\{x\}$	∃citizenOf.{USA}
minCardinalityQ	$\geq n P.C$	≥2hasChild.Lawyer
maxCardinalityQ	$\leq n P.C$	≤1hasChild.Male
cardinalityQ	$= n\, P.C$	=1 hasParent.Female

Рисунок 2.2 – DAML+OIL конструктор класів

Сенс перших трьох конструкторів (intersectionOf, unionOf та complementOf) є відносно зрозумілим: вони є лише стандартними булевими операторами, які дозволяють формувати класи з перетину, об'єднання та заперечення інших класів. Конструктор oneOf дозволяє визначати класи екзистенційно, тобто шляхом перерахування їх членів.

2.4. SPARQL запити до онтологій

SPARQL дає можливість користувачам отримувати інформацію з баз даних або будь-якого джерела даних, які можуть бути відображені в RDF. Стандарт SPARQL розроблений та затверджений W3C і допомагає користувачам і розробникам зосередитися на тому, що вони хотіли б знати, а не на тому, як організована база даних. Подібно до того, як SQL дозволяє користувачам отримувати та змінювати дані у реляційній базі даних, SPARQL забезпечує таку ж функціональність для баз даних графіків NoSQL, як GraphDB Ontotext. Крім того, запит SPARQL також може бути виконаний на будь-якій базі даних, яку можна розглядати як RDF через середнє програмне забезпечення. Наприклад, запити до реляційної бази даних можна робити за допомогою SPARQL, використовуючи програмне забезпечення для відображення реляційних баз даних RDF (RDB2RDF).

На відміну від SQL, SPARQL-запити не обмежуються роботою в одній базі даних: федеральні запити можуть отримати доступ до декількох сховищ даних (кінцевих точок). Це технічно можливо, оскільки SPARQL – це не просто мова запитів. Це також транспортний протокол на основі HTTP, де будь-яка кінцева точка SPARQL може бути доступна через стандартизований транспортний рівень. Результати RDF можна повернути в декількох форматах обміну даними, а об’єкти RDF ідентифікуються за допомогою універсальних ідентифікаторів ресурсів (URI). Підробка даних за допомогою URI дозволяє однозначно посилатись на дані в додатках та долати обмеження, що виникають при локальному пошуку. Отже, додаткові API можуть бути розроблені і можуть посилатися на ці дані. Ці варіанти дизайну - включення запитів над розподіленими джерелами на неоднорідних даних - не випадкові. SPARQL призначений для включення зв’язаних даних для семантичної мережі. Її мета - збагатити дані, пов’язуючи їх з іншими глобальними семантичними ресурсами, таким чином обмінюючись, об’єднуючись та використовуючи дані більш значущим чином [14].

Наведений нижче приклад (рисунок 2.3) відображає SPARQL-запит, визначає назву книги у заданому графі даних.

```
SELECT ?title
WHERE
{
  <http://example.org/book/book1> <http://purl.org/dc/elements/1.1/title> ?title .
}
```

Рисунок 2.3 – Приклад SPARQL-запиту

Як результат, потужність SPARQL разом із гнучкістю RDF може знизити витрати на розробку, полегшивши об’єднання результатів із кількох джерел даних.

2.5. Програмні середовища для роботи з онтологіями

На даний момент, на просторах інтернету можна знайти досить багато програмних продуктів та систем, які дозволяють проводити певні маніпуляції над онтологіями.

2.5.1. Платформа Apollo

Apollo було розроблено Knowledge Media Institute of Open University, що знаходиться в Сполученому Королівстві Великої Британії. Apollo дозволяє користувачеві моделювати онтологію з основними примітивами, такими як класи, екземпляри, функції, відносини тощо. Інтерфейс цієї програми досить простий (рисунок 2.4) і дає змогу досить зручно оперувати з онтологіями.

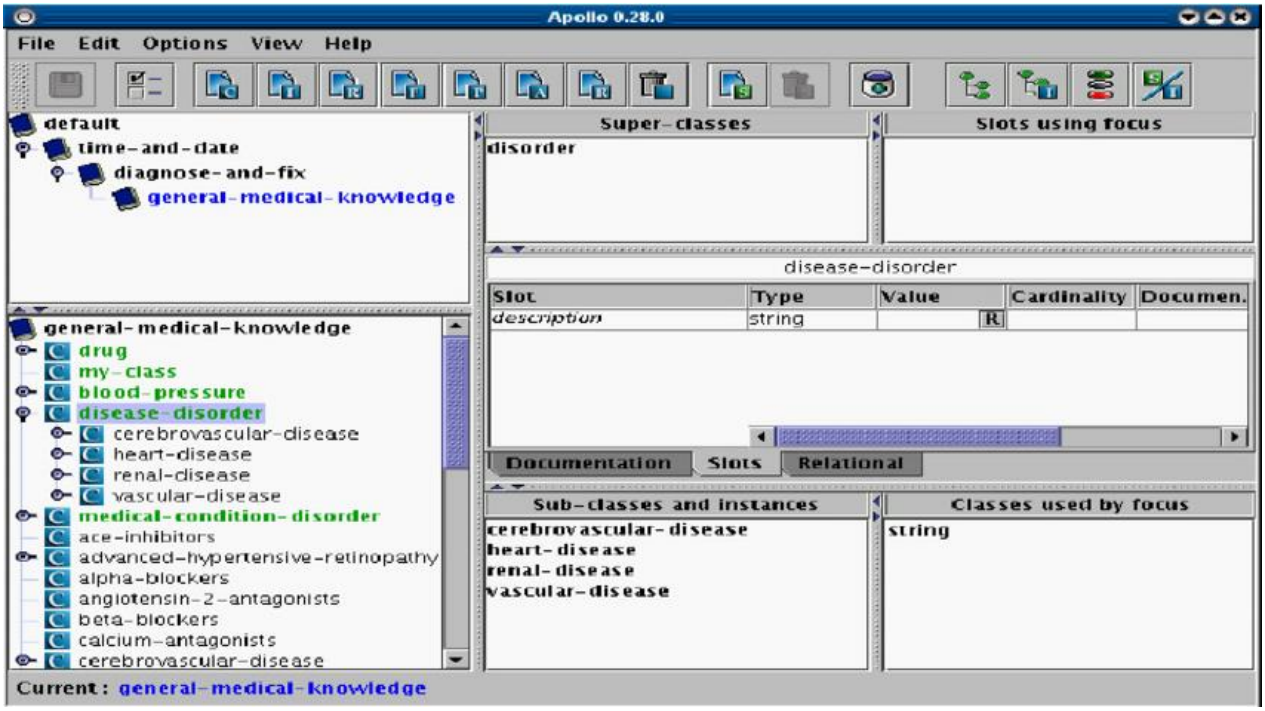


Рисунок 2.4 – Інтерфейс програми Apollo

Внутрішня модель - це кадрова система, заснована на протоколі підключення до відкритої бази знань. База знань Apollo складається з ієрархічно організованих онтологій. Онтології можуть бути успадковані від інших онтологій і можуть використовуватися так, ніби вони були власними онтологіями. Кожна онтологія має онтологію за замовчуванням, яка включає всі примітивні класи. Кожен клас може створити декілька екземплярів, а екземпляр успадковує всі слоти класу. Кожен слот складається з набору граней [15]. Apollo можна розширити за допомогою плагінів, але він не підтримує спільну роботу.

2.5.2. Редактор OntoEdit

Редактор OntoEdit розроблений компанією Ontoprise з Німеччини. Існують безкоштовні та професійні версії. OntoEdit пропонує експортні інтерфейси для всіх основних мов представлення онтологій та має гнучкий модуль для підключення. Ця функція дозволяє користувачеві налаштувати інструмент у зручному для користувача режимі. Деякі функції модулюються, тому їх можна легко розширити. На рисунок 2.5 представлено класи створеної онтології.

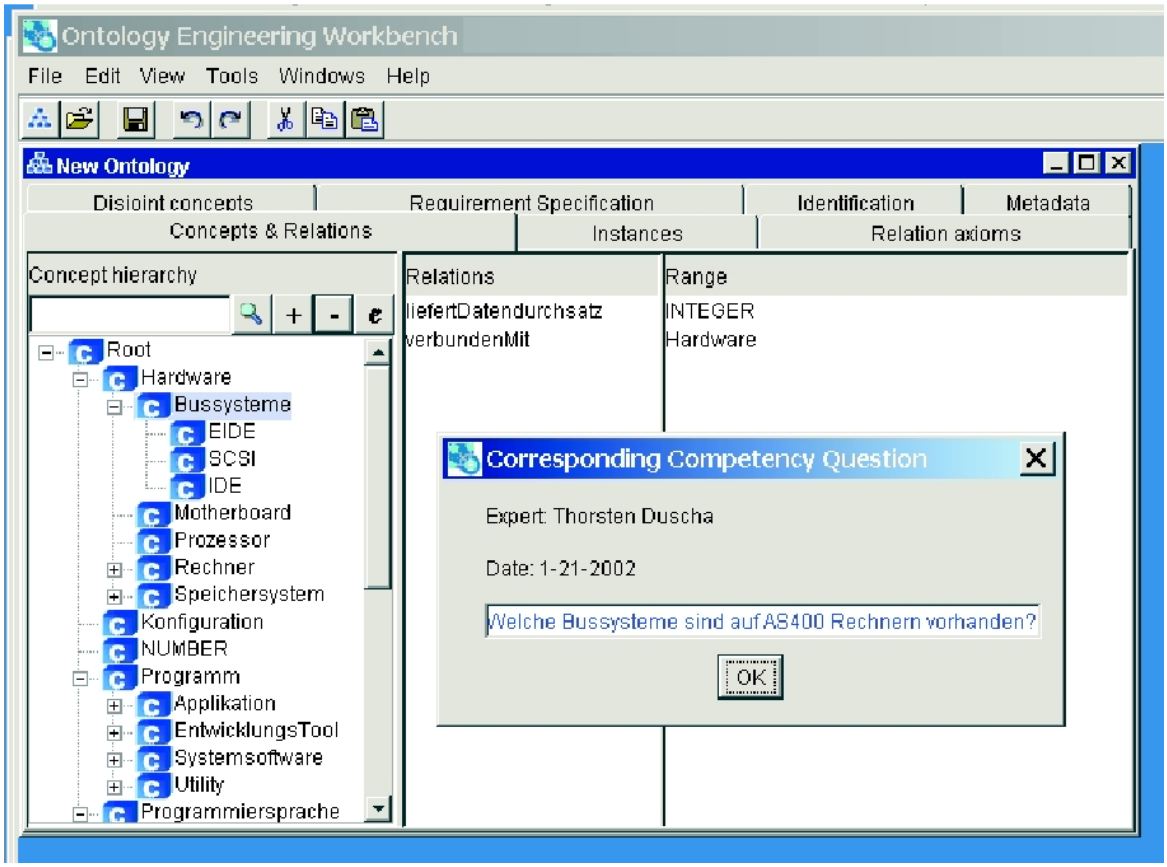


Рисунок 2.5 – Інтерфейс програми OntoEdit

Цей редактор описує документ із специфікацією онтології, в якому йдеться про те, що необхідно для підтримування та розвитку онтології. Відповідно до документа-специфікації вимог до онтології, інженер-онтолог визначає відповідні поняття та їх ієрархічну структуру в онтології. OntoEdit може бути використаний на цій фазі, використовуючи два плагіни, OntoKick та Mind20nto5, для опису метаонтології з автоматичним обчисленням статистичної інформації [15].

2.5.3. Середовище для побудови онтологій ODE

Онтології можна розглядати як доменні моделі в підході доменної інженерії. На основі онтології під час фази специфікації інфраструктури доменної інженерії можна розробити доменні інфраструктури. Для одного і того ж піддомену може бути розроблено декілька інфраструктур, кожна з яких відповідає певній технології розробки. Оскільки нас найбільше цікавить об'єктна технологія створення інфраструктури для інтеграції інструментів, ми застосували підхід для отримання об'єктних рамок із онтологій домену. Ці рамки використовуються як основа для побудови та інтеграції інструментів у SEE [16]. Для кожного інструменту або послуги, що розробляється, специфікація вимоги повинна бути виконана у підході з повторним використанням, тобто, використовуючи та спеціалізуючи отримані рамки (рисунок 2.6). Це підхід, який використовується в ODE (середовищі розробки програмного забезпечення, заснованого на онтології), SEE, орієнтованому на процеси.

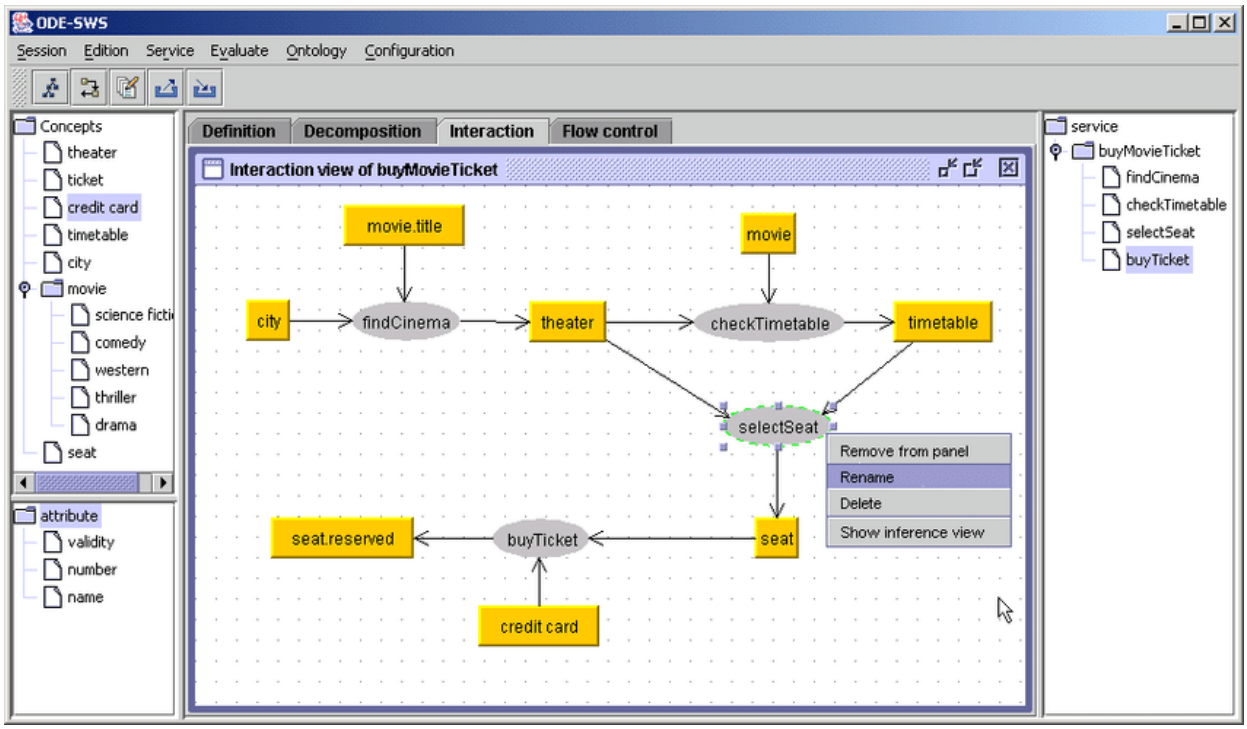


Рисунок 2.6 – Інтерфейс програми ODE

Оскільки SEE можна розглядати як (часткову) автоматизацію програмного процесу, ядро процесу ODE будується на основі програмного процесу, що випливає з

онтології програмного забезпечення. Інструменти для визначення процесів та відстеження проектів також будуються на цій основі [17].

2.5.4. Програма OilEd

OilEd розроблялася групою з управління інформацією відділу CS при Університеті Манчестера, Великобританія. OilEd – це OlLeditor, який дозволяє користувачеві створювати та редагувати онтології OIL. OilEd в першу чергу призначений для демонстрації використання DAML + OIL, але він не підтримує повне середовище для розвитку онтології [15]. Хоча, попри це нею досить зручно будувати класову ієрархію онтології (рисунок 2.7).

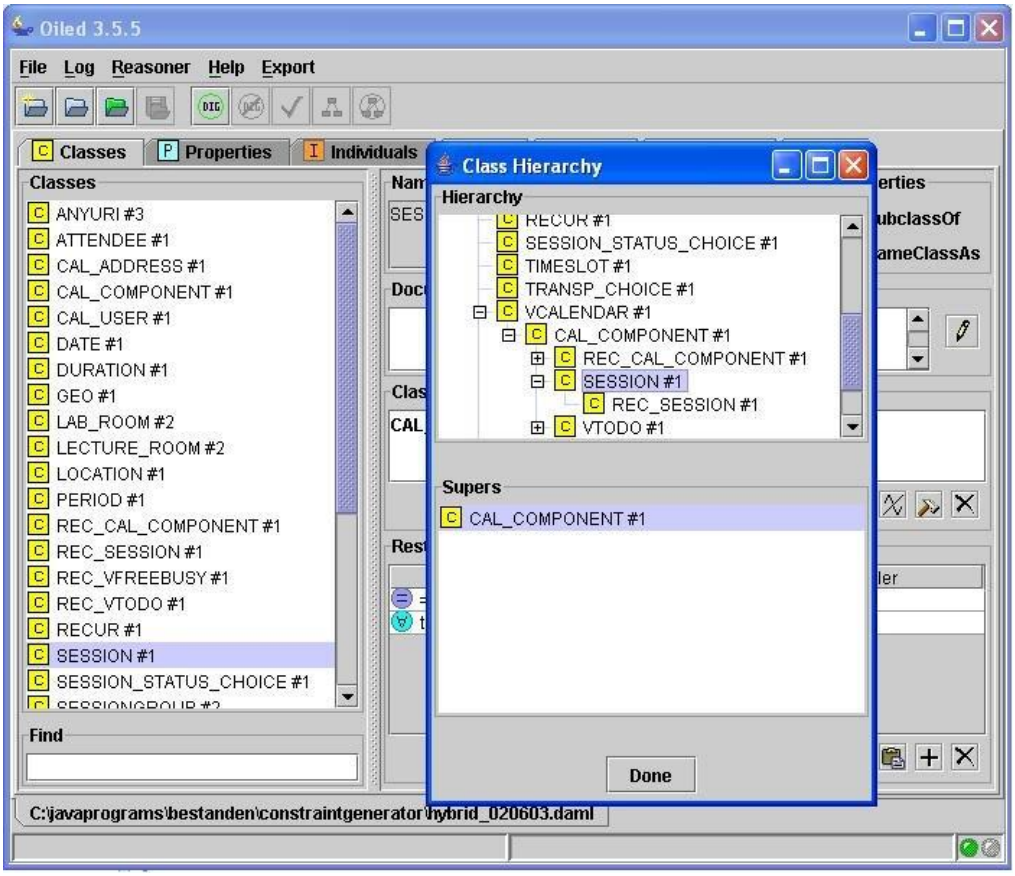


Рисунок 2.7 – Інтерфейс програми OilEd

OilEd не підтримує багатьох видів діяльності, таких як створення широкомасштабних онтологій, модифікація версій, збільшення, міграція та інтеграція онтологій, які беруть участь у побудові онтології. OilEd не має розширюваності, але

можна використовувати довільні вирази класів, примітивні та визначені класи та вирази конкретного типу.

2.5.5. Редактор OntoStudio

OntoStudio – це професійне середовище онтологій, що розвивається. Воно поєднує інструменти моделювання онтологій та правил з компонентами для інтеграції різнорідних джерел даних. Завдяки своєму модульному дизайну, OntoStudio можна збагатити самостійно розробленими модулями та налаштувати під особисті потреби. Як онтологічні мови, OntoStudio підтримує W3C-стандарти OWL, RDF і RDFS, і F-Logic для логічної обробки правил. OntoStudio поставляється разом з багатьма роз'ємами до баз даних, документів, файлових систем, додатків та веб-служб. Інтерфейс даного застосунку представлено на рисунок 2.8.

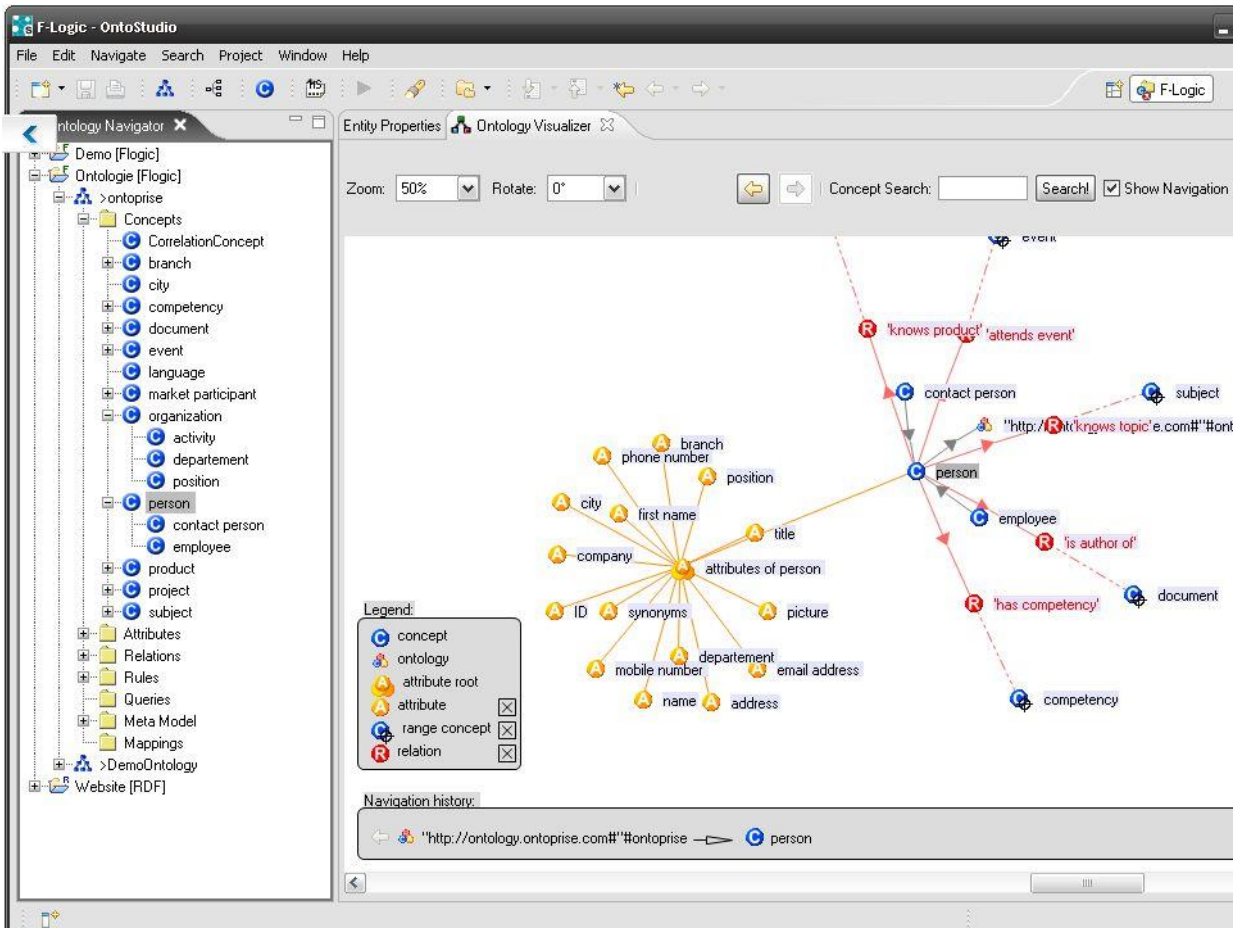


Рисунок 2.8 – Інтерфейс програми OntoStudio

Однією із найбільш популярних версій цієї програми є OntoStudio X, що являє собою радикальну гібридну архітектуру IDE, яка відповідає всім поточним та майбутнім потребам інтелектуальної та оперативної обробки даних. На основі Microsoft Excel 2019 як оболонки адміністрування, мови онтології, декларативні функціональні програми та інтерфейси процедурних програм (Java, Python) пов'язані між собою у високій продуктивності. Наприклад, нейронні алгоритми ML, реалізовані в Python, можуть відображати свої результати безпосередньо в онтологіях HOL для подальшої семантичної перевірки правдоподібності [18].

2.5.6. Програма WebOnto

WebOnto є онтологічним редактором онтологій OCML (Мова оперативного концептуального моделювання) і був розроблений в Knowledge Media Institute of Open University. Цей інструмент є аплетом Java в поєднанні з налаштованим веб-сервером і дозволяє користувачам переглядати та редагувати моделі знань через Інтернет. Найбільшою перевагою цього редактора є побудова графів онтологій (рисунок 2.9).

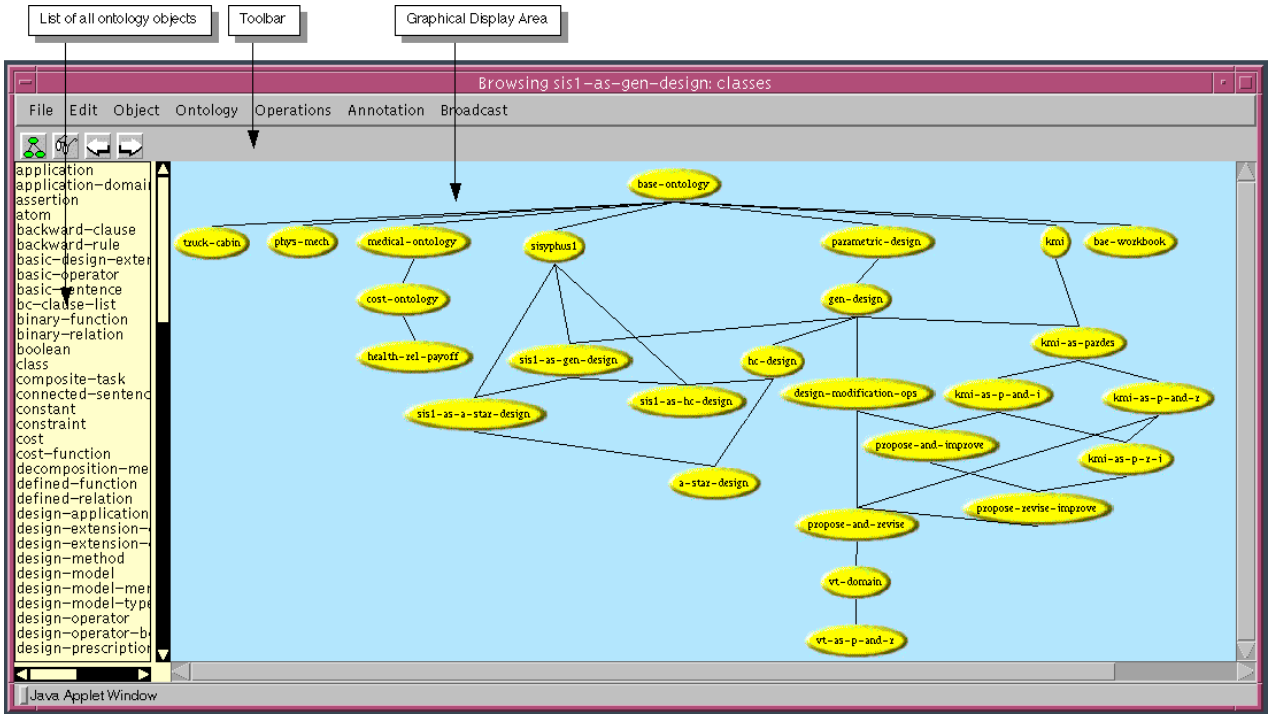


Рисунок 2.9 – Інтерфейс програми WebOnto

Факт, що WebOnto зміг підтримати спільне редагування онтології, було головною перевагою на той час. Ontolingua, Ontosaurus та WebOnto були створені виключно з метою перегляду та редагування онтологій на певній мові (Ontolingua, LOOM та OCML відповідно). Цих редакторів старшого покоління навряд чи можна було розширити порівняно з інженерними середовищами сьогодні. Нове покоління онтологічно-інженерних середовищ є більш досконалим та амбітним, ніж їх попередники [19]. Вони є розширюваними, мають складові архітектури, де нові модулі можна легко додати, щоб забезпечити більше функціонального середовища.

2.6. Висновки до розділу

В даному розділі було досліджено предметну область для побудови майбутніх онтологій, а також розглянуто мови, за допомогою яких можна оперувати онтологіями. Більше того, було описано програмні продукти та редактори для роботи з онтологіями та досліджено переваги та недоліки кожного.

3.ЗАСОБИ РОЗРОБКИ ПРОГРАМНОГО ЗАСТОСУНКУ

Під час розробки програмного застосунку були задіяні такі середовища коду та додатки:

- програма для створення та редагування онтологій Protégé 5.5.0;
- середовище для розробки коду програмного продукту Visual Studio 2017;
- програмна платформа .Net Framework;
- бібліотека RDF для .Net Framework для роботи з онтологіями засобами мови C#;

Перш за все, спочатку необхідно створити каркас – онтології для зберігання даних про навчальні ресурси кафедри. Потім у середовищі для розробки програмного коду Visual Studio 2017 засобами мови C# створити клієнтський застосунок, який дозволить клієнту виконувати запити до програми та отримувати необхідні дані.

3.1. Програма Protégé 5.5.0

Protege - редактор онтології з відкритим кодом. Онтологія схожа на систематику тим, що вона представляє контрольований словниковий запас для даної області знань. Однак відносини між різними об'єктами можуть бути набагато складнішими та більш описаними. Це дозволяє користувачам створювати онтології як в рамках Frames, так і в веб-онтологічній мові (OWL). Protege дозволяє користувачам робити

Protege призначений для тих, хто працює в галузі онтології та моделювання знань, оскільки майже завжди потрібен певний ступінь знань про основні аксіоми. Деякі доступні плагіни, які певною мірою захищають користувача від них.

Для працюючих вчених найбільш корисними плагінами та видами будуть ті, які графічно представляють складні моделі знань. Крім того, це може бути корисно тим, хто бажає використовувати вираз RDF / XML Dublin Core для анотації своїх даних метаданими. Існують численні плагіни, написані іншими проектами. Інтерфейс (рисунок 3.1) та принцип роботи цієї програми досить простий.

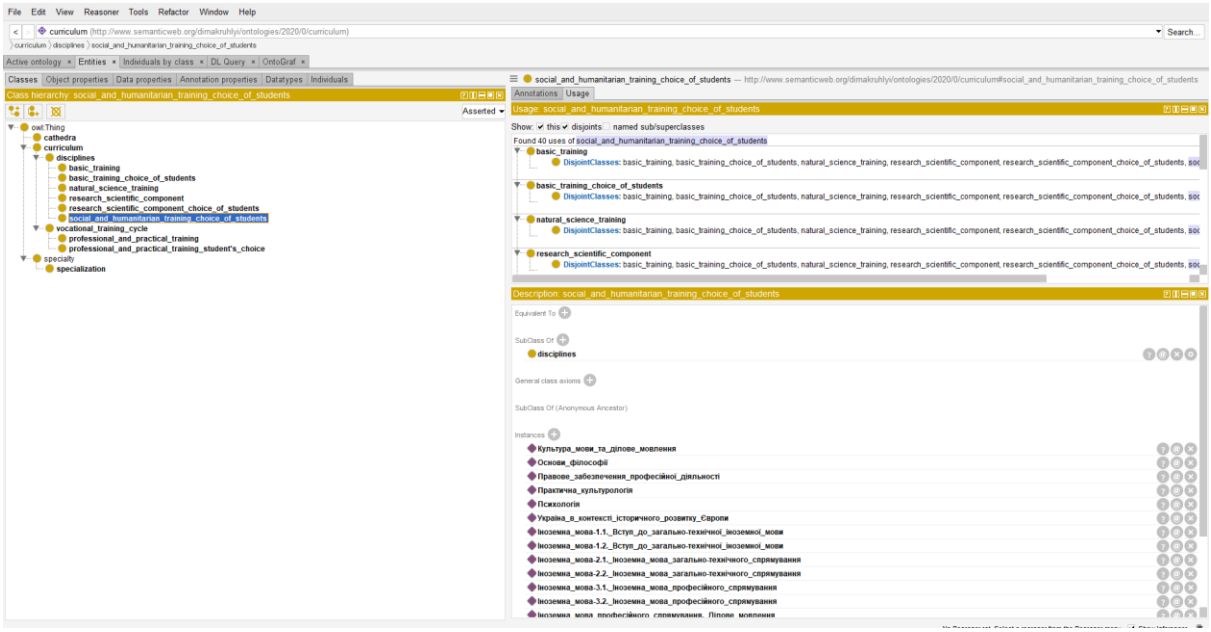


Рисунок 3.1 – Інтерфейс програми Protégé 5.5.0

Protege можна використовувати для редагування більш простих словникових систем, таких як Схема простої організації знань (SKOS), але загалом її потужність є надмірною для цього використання. Protégé підтримується сильною спільнотою академічних, державних та корпоративних користувачів, які використовують цю програму для побудови рішень, заснованих на знаннях, у таких різноманітних сферах, як біомедицина, електронна комерція та організаційне моделювання [20]. Плагінна архітектура Protégé може бути адаптована для створення простих і складних програм на основі онтології. Розробники можуть інтегрувати вихід Protégé із системами правил або іншими рішеннями проблем для побудови широкого спектру інтелектуальних систем.

3.2. Середовище Visual Studio 2017

Інтегроване середовище розробки Visual Studio - це креативна панель запуску, яку можна використовувати для редагування, налагодження та створення коду та публікації програми. Інтегроване середовище розробки (IDE) - це багатофункціональна програма, яка може бути використана для багатьох аспектів розробки програмного забезпечення. Окрім стандартного редактора та налагоджувача, який надає більшість IDE, Visual Studio включає компілятори, інструменти для завершення коду, графічні дизайнери та багато інших функцій для полегшення процесу розробки програмного забезпечення.

Час від часу Microsoft вносила багато змін із кожною версією цього IDE. Перед Visual Studio 2017 RC Microsoft випустила Visual Studio 2017 RC, і вона була дуже популярною серед програмістів. Нарешті вийшов Visual Studio 2017. У цьому середовищі стільки нових функцій порівняно з Visual Studio 2015, що допомагає розробникам дуже легко створювати, розгортати та налаштовувати будь-які типи програм. Редактор не тільки пропонує нові функції (рисунок 3.2), але також, він інтегрує різні пакети в одному місці. Він доставить розробникам .NET Development, Cloud, Mobile, Linux та багато інших досвіду розробки.

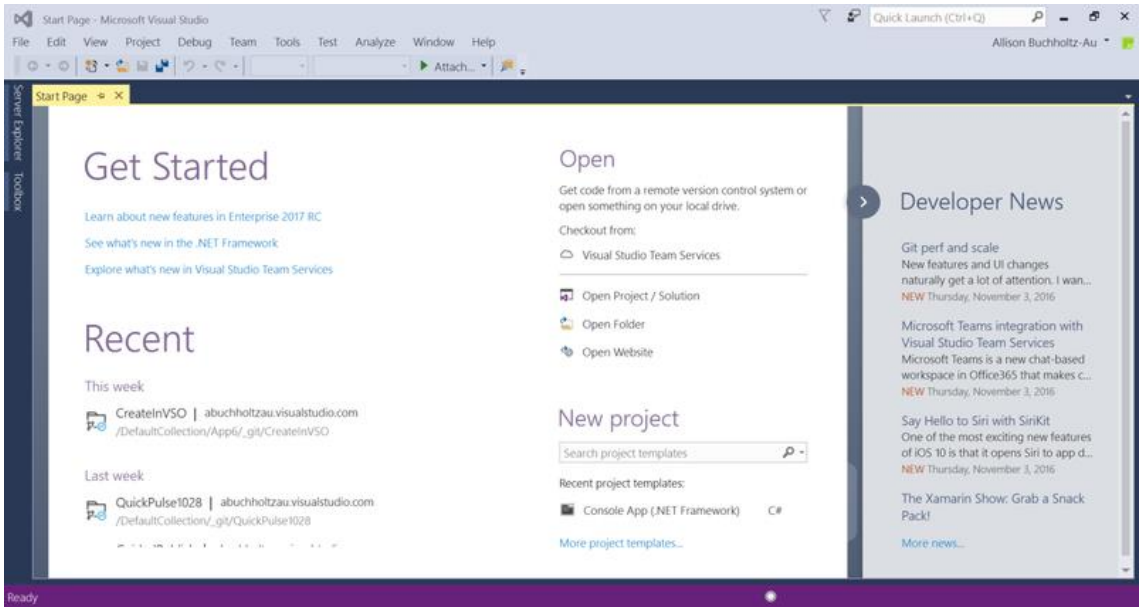


Рисунок 3.2 – Start page середовища Visual Studio 2017

Xamarin тепер є частиною Visual Studio 2017 і інтегрується з Visual Studio 2017. Це чудова новина для розробників або програмістів, які працюють над розробкою мобільних додатків, таких як Android, iOS та Windows, і готові працювати з чудовим IDE [21]. Тепер, за винятком Visual Studio, не можна запустити цей IDE. Дуже велика кількість плагінів цього інтегрованого середовища для створення клієнтських (і не тільки) програм суттєво полегшує написання коду програмістами. Також кожен розробник може вибрати тему оформлення коду, яка йому до вподоби (рисунок 3.3).

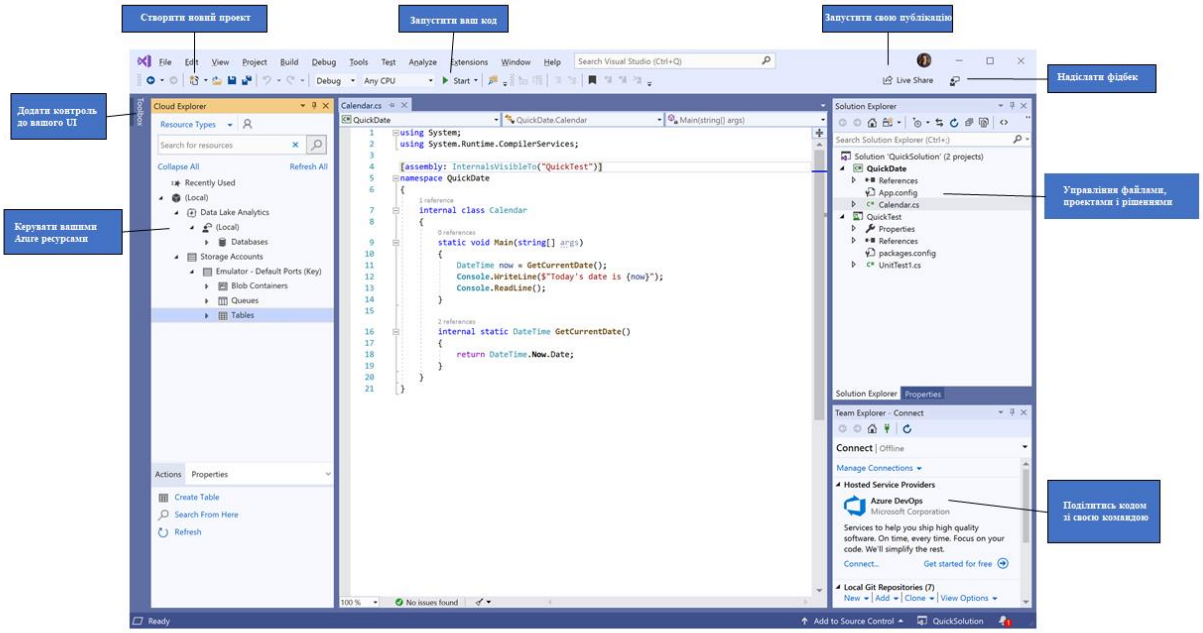


Рисунок 3.3 – Інтерфейс для написання коду у Visual Studio 2017

Visual Studio – це не лише чудовий інструмент, але він також пропонує багато нових додаткових функцій. Зараз це набагато швидше порівняно з попередніми версіями. .NET Core та Cross Platform Development, які були довгоочікуваними особливостями Visual Studio, прийшли разом із Visual Studio 2017.

3.3. Програмна платформа .Net Framework

Якщо ви дуже довго використовуєте Windows, ви, мабуть, чули про Microsoft .NET, можливо, через те, що програма попросила вас встановити її чи ви помітили її у своєму списку встановлених програм. Якщо ви не розробник, вам не потрібно багато знань, щоб скористатися ним. Вам це просто потрібно для роботи. Назва «.NET Framework» сама по собі є дещо помилкова. «Framework» (з точки зору програмування) - це справді сукупність інтерфейсів програмування прикладних програм (API) та спільної бібліотеки коду, яку розробники можуть викликати при розробці програм, так що їм не доведеться писати код з нуля. У .NET Framework ця бібліотека спільного коду називається бібліотекою Framework Class (FCL). Біти коду в спільній бібліотеці можуть виконувати всілякі різні функції [22]. Скажімо, наприклад, розробнику потрібна їх програма, щоб мати можливість записувати іншу IP-адресу в мережі. Замість того, щоб писати сам цей код, а потім записати всі дрібні шматочки та фрагменти, які повинні інтерпретувати, що означають результати ping, вони можуть використовувати код з бібліотеки, яка виконує цю функцію (рисунок 3.4).

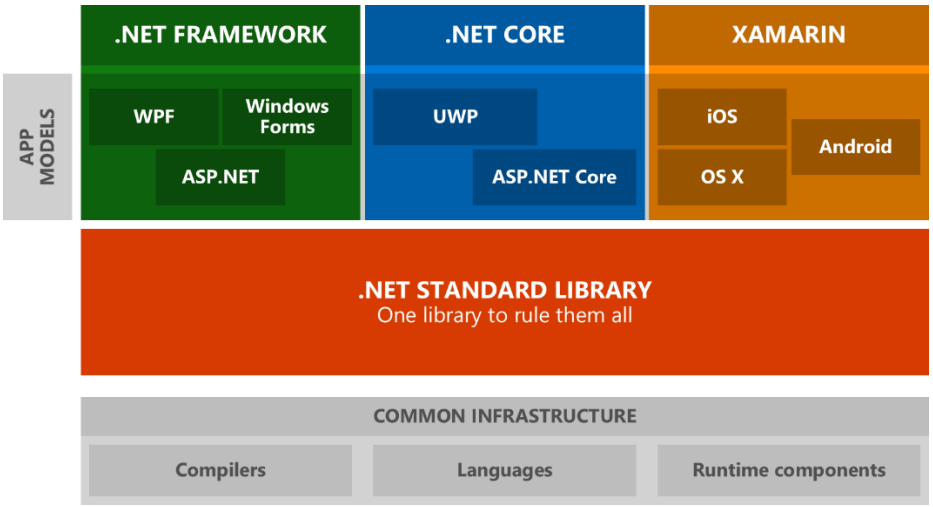


Рисунок 3.4 – Структура .Net Framework

І це лише один крихітний приклад. .NET Framework містить десятки тисяч частин спільного коду. Цей спільний код значно полегшує життя розробників, оскільки їм не

доведеться винаходити колесо кожного разу, коли їх додатки повинні виконувати якусь загальну функцію. Натомість вони можуть зосередитись на коді, який є унікальним для їхніх програм та на користувальницький інтерфейс, який пов’язує все це разом. Використання подібного фреймворку також допомагає забезпечити деякі стандарти між програмами. Інші розробники можуть зрозуміти, що програма робить легше, і користувачі програм можуть розраховувати на такі речі, як діалогові вікна "Відкрити і зберегти як", які працюють однаково в різних додатках.

Оскільки, крім того, що служить основою спільного коду, .NET також забезпечує середовище виконання програм. Середовище виконання забезпечує віртуальну машинну пісочницю, в якій запускаються програми. Багато платформ розробки пропонують таку ж річ. Наприклад, Java та Ruby on Rails надають власні умови виконання. У світі .NET середовище виконання називається загальною мовою виконання (CLR). Коли користувач запускає додаток, код цього додатка фактично компілюється в машинний код під час виконання і потім виконується. CLR також надає деякі інші послуги, такі як управління пам’яттю та потоками процесора, обробка винятків програми та управління безпекою [22]. Середовище виконання дійсно є способом абстрагування програми від фактичного обладнання, на якому працює програма.

3.4. Бібліотека RDF для .Net Framework

dotNetRDF - це бібліотека .Net, написана на C #, створена для надання простого, але потужного API для роботи з даними опису ресурсів (RDF). В цілому, вона пропонує велику різноманітність класів для виконання всіх загальних завдань - від читання та запису RDF-даних до запитів над ними. Бібліотека розроблена таким чином, щоб вона була дуже розширюваною і дозволяла користувачам додавати підтримку додаткових функцій (наприклад, користувальницькі потрійні магазини RDF) за потребою.

Бібліотека функціонує в основному на рівні Triples, Graphs та Triple Stores та забезпечує дуже обмежену підтримку Inference та відсутність прямої підтримки OWL.

Основні класи бібліотеки можна знайти в просторі імен VDS.RDF. Усі основні класи засновані або на інтерфейсах, або на абстрактних класах, щоб зробити бібліотеку максимально розширюваною [23]. Документ RDF можна вважати таким, що формує математичний граф, і тому ми представляємо набори потрійних структур у вигляді графів. Усі графи в бібліотеці є реалізацією інтерфейсу IGraph (рисунок 3.5) і, як правило, походять від абстрактного класу BaseGraph, який реалізує деякі основні методи інтерфейсу, що дозволяє певним реалізаціям концентруватися на таких характеристиках, як стійкість до зберігання / безпеки потоку. Реалізація IGraph - це представлення в пам'яті документа RDF. Найпоширенішою реалізацією IGraph є клас Graph.

```
//Get a new Graph and set it's Base URI
IGraph g = new Graph();
g.BaseUri = new Uri("http://example.org/");
```

Рисунок 3.5 – Створення об’єкту класу IGraph

Триплети можуть бути додані до Graph методом Assert (...), метод займає один Triple або масив / список / перелік Triples [24]. Після додання до Graph деяких даних, їх можна перебрати використовуючи цикл foreach (рисунок 3.6).

```
//Loop through Triples
foreach (Triple t in g.Triples)
{
    Console.WriteLine(t.ToString());
}
```

Рисунок 3.6 – Перебір даних за допомогою foreach

Будь-який IGraph реалізує IEnumerable <Triple> і тому може бути використаний з усіма методами розширення LINQ для IEnumerable <T>.

3.5. Висновки до розділу

В даному розділі було описано програмний продукт для розробки онтологій предметної області, а також було розглянуто середовище та інструменти для написання коду клієнтського застосунку. Також було наведено прості приклади коду для роботи з бібліотекою RDF за допомогою платформи .Net Framework.

4. ОПИС ПРОГРАМНОЇ РЕАЛІЗАЦІЇ

Даний розділ описує процес покрокового створення онтологій та заповнення їх відповідними даними. Основною частиною дипломної роботи являється створення правильних онтологічних файлів формату .owl, в яких і будуть міститься всі необхідні дані про навчальний процес кафедри. На рисунок 4.1 зображено структурну схему системи.

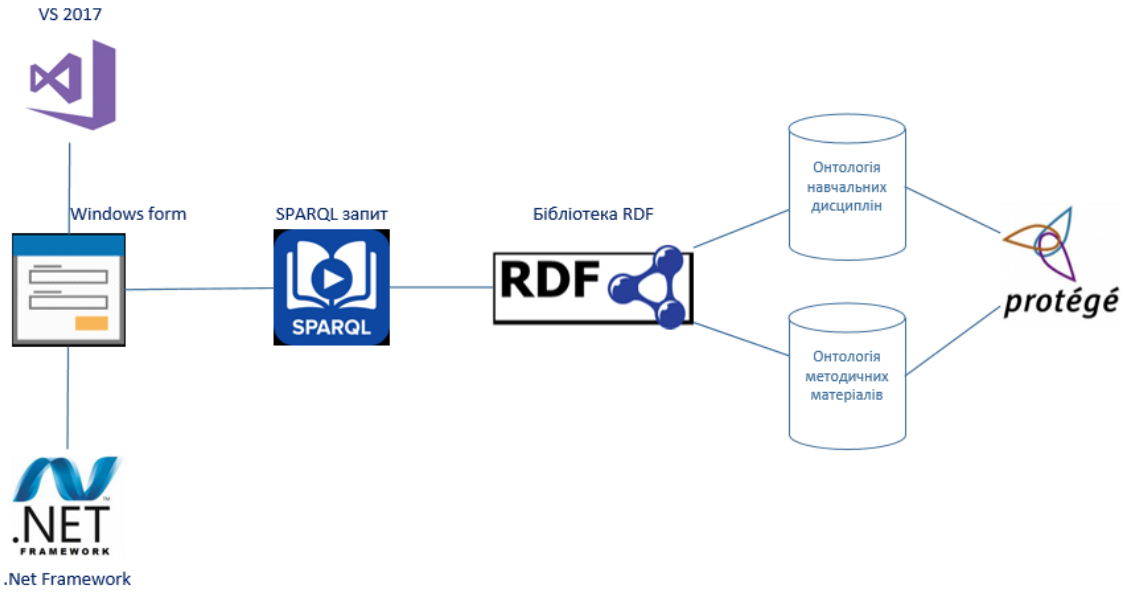


Рисунок 4.1 – Структурна схема системи

Базою даних тут слугують два онтологічні файли формату .owl, які були створені у програмі Protégé. Інтерфейс користувача написано з використанням .Net Framework та середовища розробки Visual Studio 2017. Користувач взаємодіючи з програмою відправляє SPARQL запит, який обробляється бібліотекою RDF для фреймворку .Net.

Далі буде розглядатися структура і необхідність написання SPARQL запиту до отриманих онтологій, а також пов’язання всіх елементів системи в один єдиний програмний продукт.

4.1. Загальний опис системи

Майбутня створювана система буде складатися з двох онтологічних файлів, як уже було примічено раніше, формату .owl. Під час розробки постала необхідність саме в двох файлах онтології замість одного, перш за все, тому що обсяг даних, з яким доведеться працювати в подальшому і який з кожним роком буде постійно накопичуватися є досить великим і створювати одне дерево класів для всієї інформації досить обтяжливо. Тому було прийнято рішення створити два окремих файли з онтологіями по визначеній предметній області. Перший файл буде містити всю необхідну інформацію про методичні матеріали, нормативні документи, файли, книги та інші ресурси. В другому ж файлі буде знаходитися інформація про предмети, що викладаються на кафедрі, кількість годин, кредитів, сортування предметів по семестрах та навчальних роках.

В цілому, можна виділити дві ролі, які працюють із даною системою – користувач і розробник. На рисунок 4.2 відображено взаємодію користувача з даною системою.

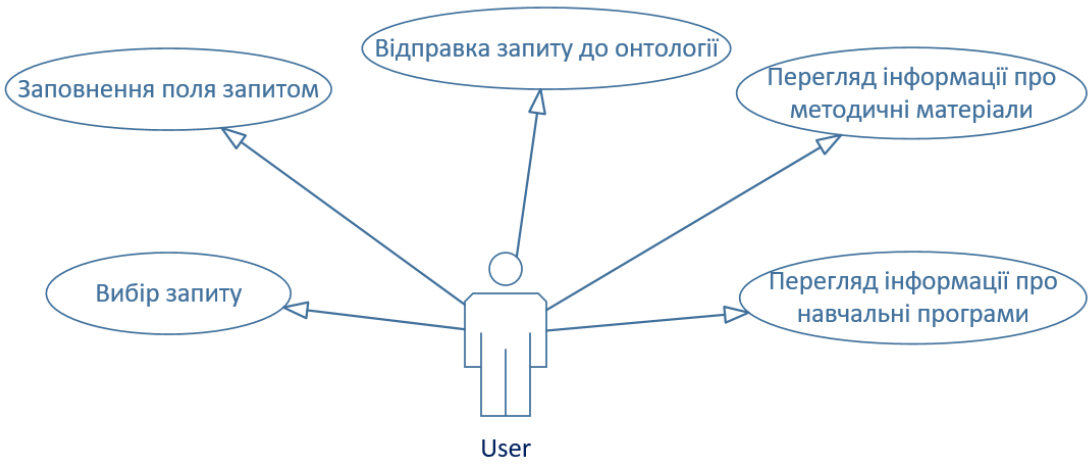


Рисунок 4.2 – Взаємодія користувача з даною системою

З цієї схеми видно, що навіть користувач, який не має спеціальних знань в програмуванні чи роботі з онтологіями, може з легкістю експлуатувати дану систему, відправляючи SPARQL запити до створених онтологій.

Другою ж особою, яка має безпосереднє відношення до даної системи є розробник. Спектр його можливостей значно ширший, ніж у звичайного користувача. На рисунок 4.3 представлено «арсенал» розробника.

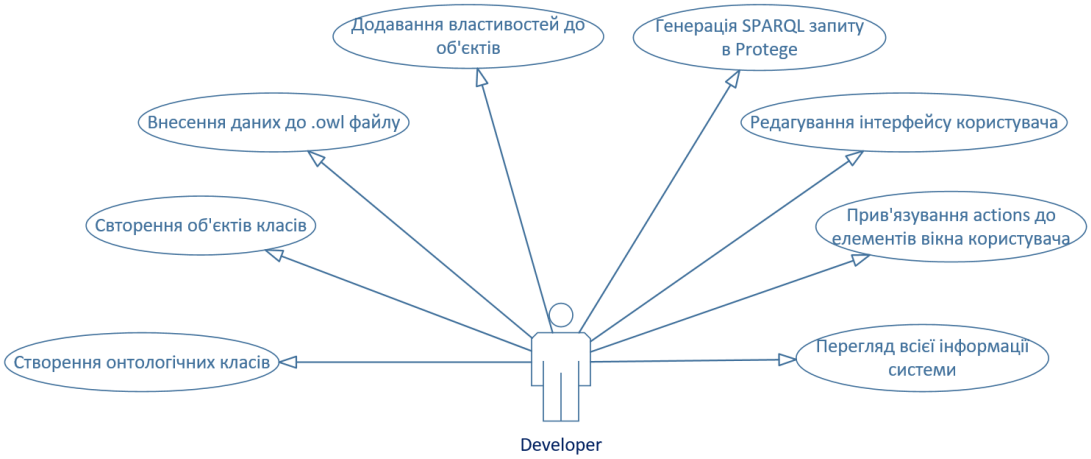


Рисунок 4.3 – Взаємодія розробника з системою

З поданих вище діаграм ми можемо бачити, що система є досить оптимальною, адже, щоб отримати певний результат, не потрібно вдумуватися в те, як працює алгоритм пошуку даних або за якими критеріями побудований запит до онтологій.

4.2. Створення онтологій

Створення будь-якої онтології базується на побудові класової системи, створенні зв’язків між ними та реалізація екземплярів класів. Після чого йде заповнення заданої системи відповідними даними. Так, як маємо дві онтології, то доцільніше буде розглянути кожну з них по порядку.

Першою пропонується розглянути онтологію про навчальні предмети та планування. Основними класами даної онтології є «cathedra», «curriculum» та

«speciality». Вони включають в себе інші підкласи, які в свою чергу мають свої підкласи (рисунок 4.4).

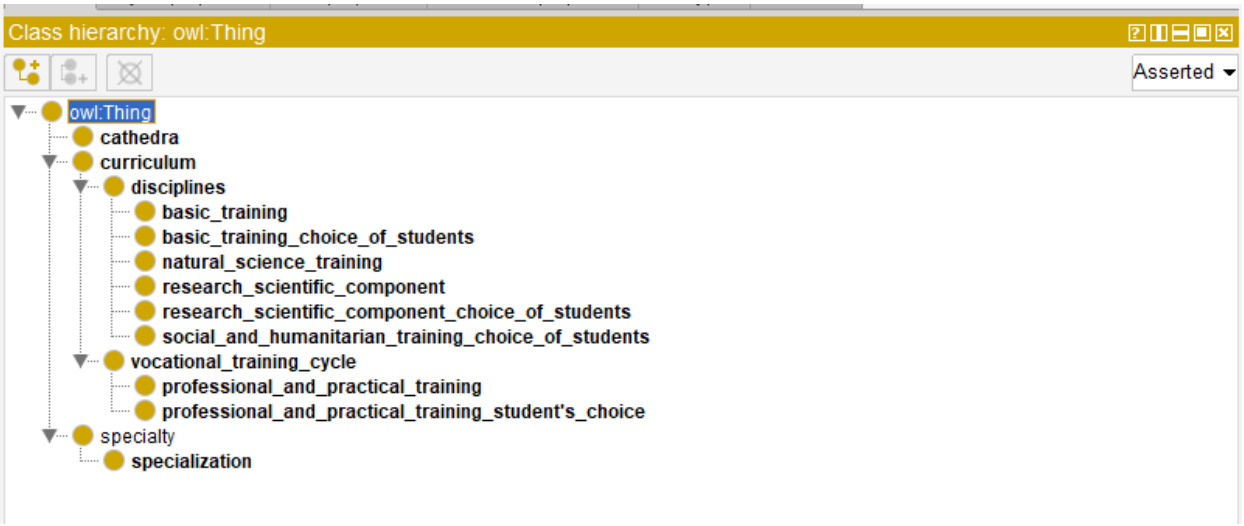


Рисунок 4.4 – Ієрархія класів онтології навчальних дисциплін

Також в Protégé 5.5.0 є чудовий інструмент OntoGraf, який дозволяє нам переглянути дану ієрархію класів у вигляді графу (рисунок 4.5).

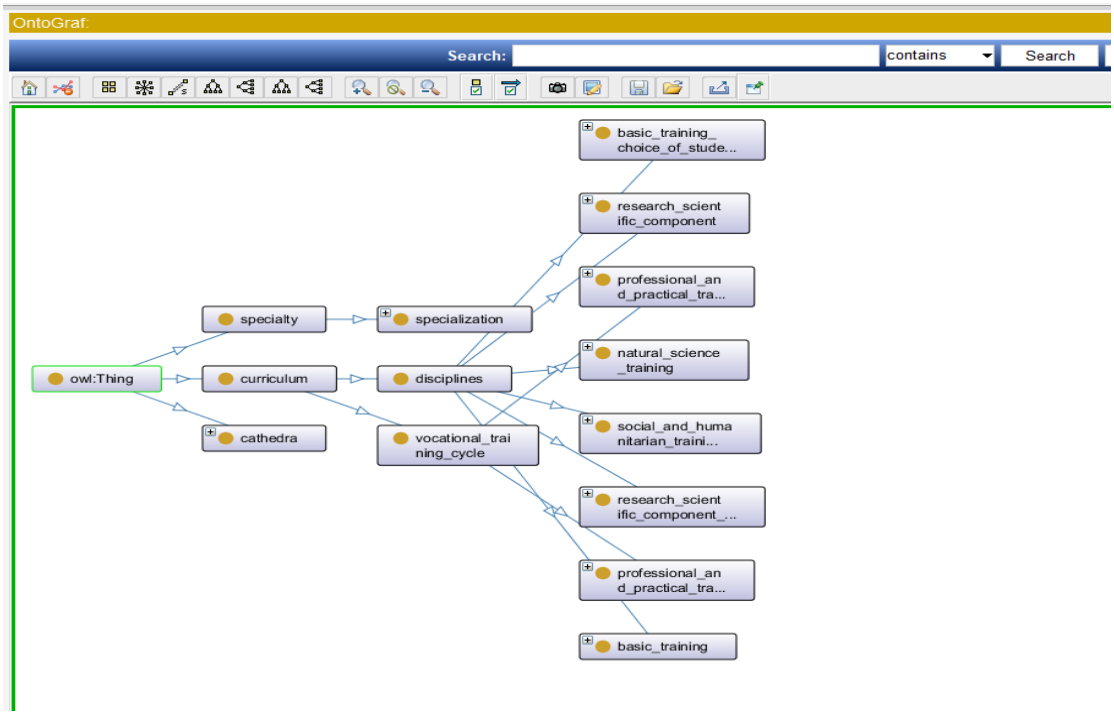


Рисунок 4.5 – Представлення ієрархії класів у вигляді графа

У Protégé також можна задавати властивості для екземплярів класу. Всі властивості діляться на дві групи:

- object properties (властивості об’єктів);
- data properties (властивості даних).

В основному, властивостями об’єктів є відносини між двома і більше екземплярами класу. За замовчуванням, так як і під час побудови ієрархії класів було створено клас «Thing», то під час додавання object properties за замовчуванням уже існує властивість «topObjectProperty». Екземпляри класу можуть містити властивості, які задано на рисунок 4.6. Якщо розглянути цей рисунок більш детально, то можна помітити схожість між властивостями «belong» і «has», а також між «is_taught» і «teaches». Вони схожі між собою, про те використовуються для різних екземплярів.

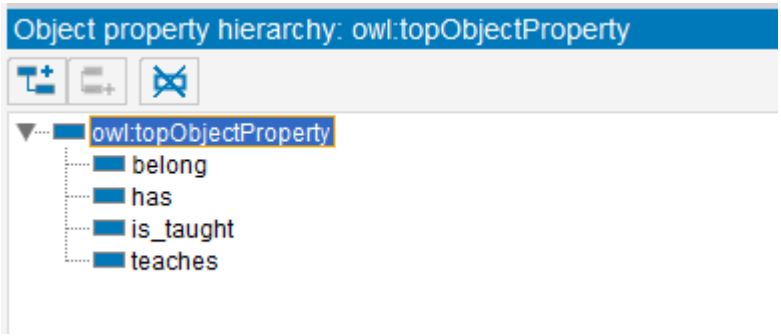


Рисунок 4.6 – Представлення ієрархії властивостей об’єктів

Другий тип властивостей – це data properties. Основна відмінність від object properties - це те, що вони описують зв’язок між екземпляром та значенням даних. При додаванні нових властивостей даних також за замовчуванням створено властивість «topDataProperty». Властивості даних описують, які значення будуть міститься в екземплярах, створених від певних класів (рисунок 4.7).

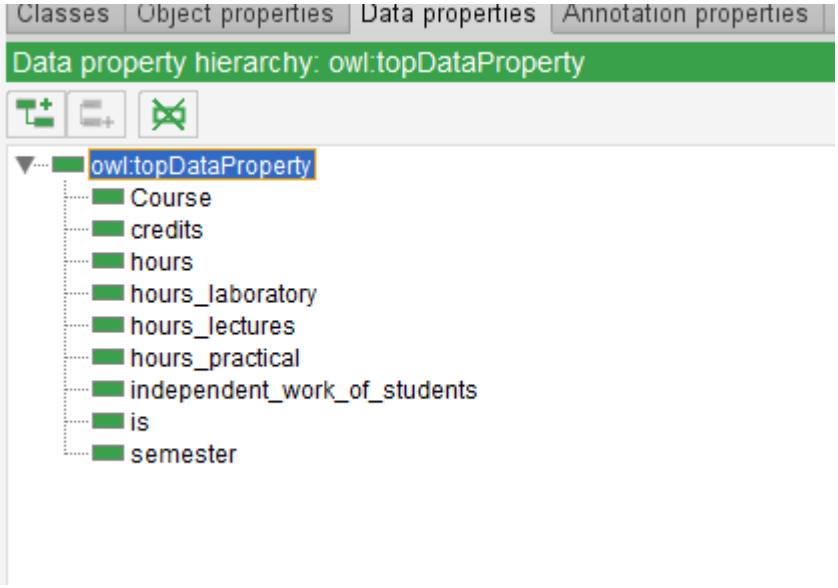


Рисунок 4.7 – Представлення ієрархії властивостей даних

Важливість даних властивостей полягає в тому, що при створенні об’єкта певного класу, вони автоматично є доступними для нього. Потрібно лише вибрати необхідні властивості і прив’язати їх до об’єкта (рисунок 4.8).

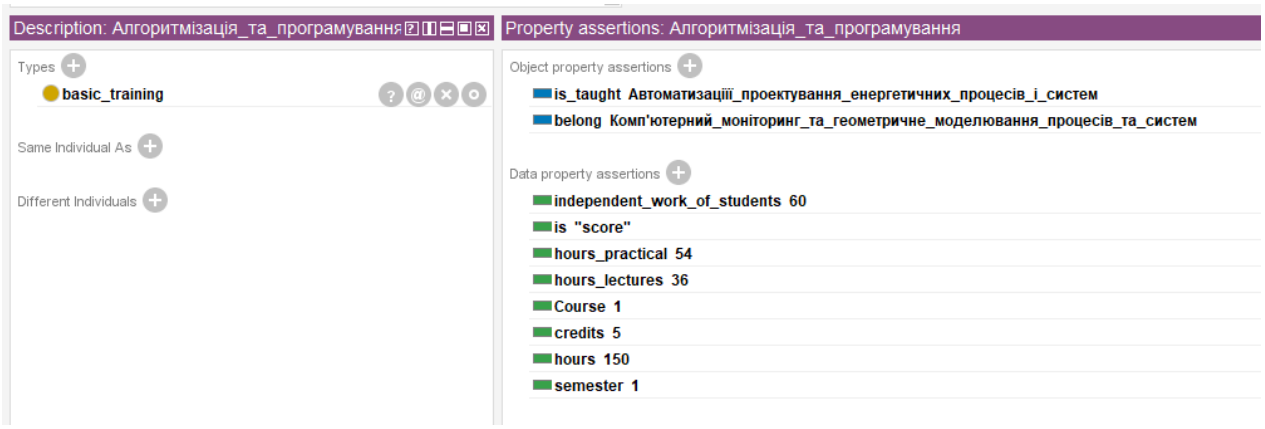


Рисунок 4.8 – Задання властивостей об’єкту «Алгоритмітизація та програмування»

Про те, основним недоліком редактору онтологій Protégé є те, що всі дані в ньому знаходяться в одному місці (рисунок 4.12). Працювати з ними в такому форматі досить незручно.

Тепер неохідно детальніше розглянути другу побудовану онтологію. Вона містить інформацію про методичні матеріали, документи, посібники кафедри. Так, як

найголовнішими критеріями пошуку документів було вибрано спеціальність та викладача, то маємо два основні класи «Speciality» та «Teachers». Кожен з цих класів містить свої підкласи, а ті в свою чергу мають свої підкласи (рисунок 4.10) і тд.

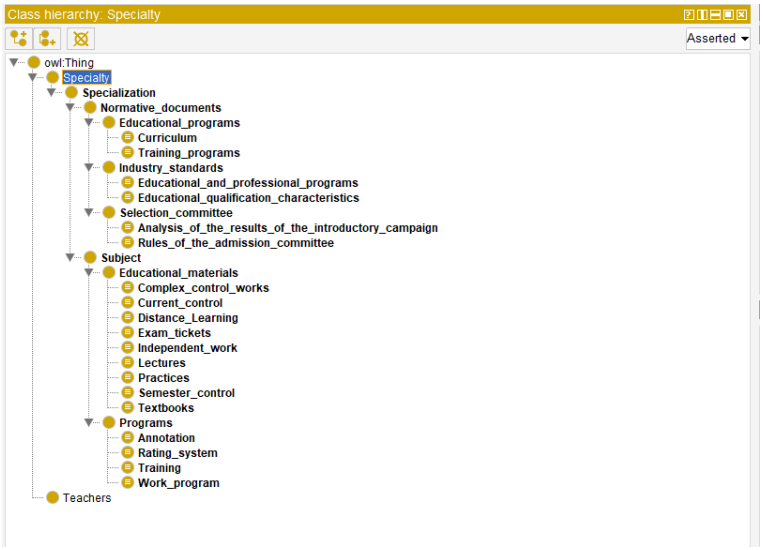


Рисунок 4.9 – Ієрархія класів онтології методичних матеріалів

Як було згадано раніше, всі властивості діляться на два види. На рисунок 4.10 зображено object properties для екземплярів класів. Властивість «has_author» буде ссилатися на викладача, який є автором певного документу. Що ж стосовно властивостей «refers_to_speciality» та «refers_to_subject», то вони ссилаються на спеціальність та предмет, відповідно, до яких належать документи, а також за яким будуть здійснюватися SPARQL запити до онтології (рисунок 4.11).

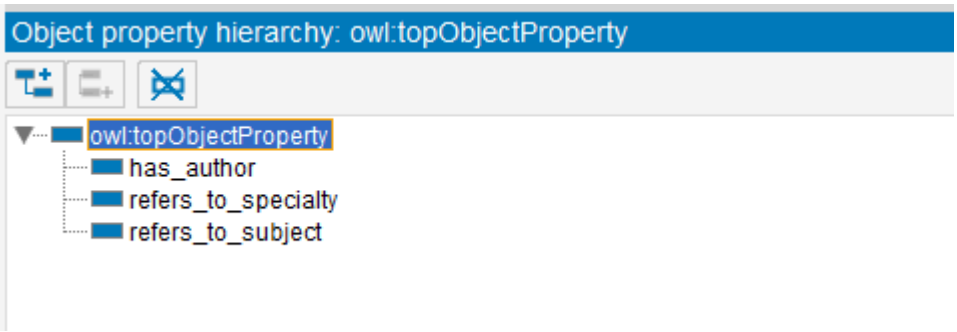


Рисунок 4.10 – Представлення ієрархії властивостей об’єктів

Властивості даних описують же курс, на якому викладається даний предмет і необхідно методичний матеріал, кількість примірників, мова написання, тип, назва та тому подібне (рисунок 4.12).

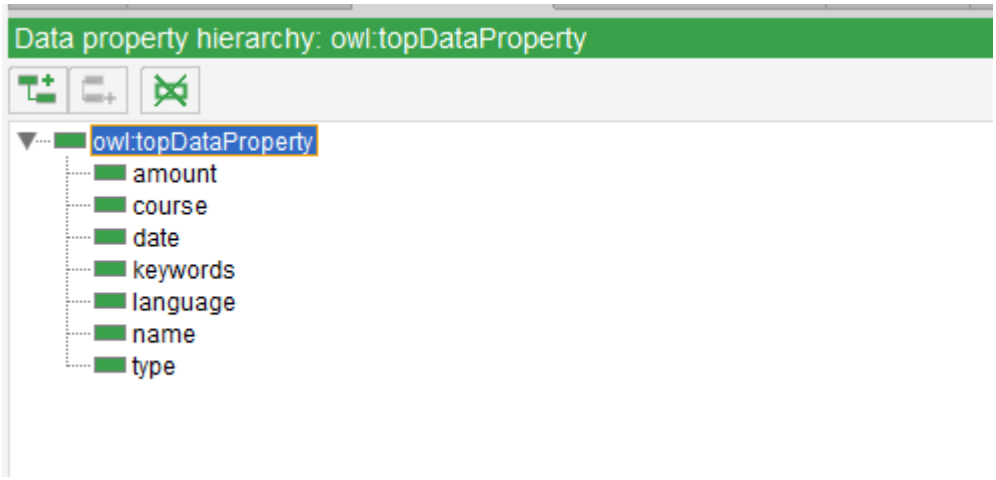


Рисунок 4.11 – Представлення ієрархії властивостей даних

Створення екземплярів класів та заповнення онтології даними відбувається у тих самих вкладках та полях, які описувалися декількома абзацами вище. Для більшої наглядності на рисунку 4.12 зображено заповнення екземплярів класів даними із вкладки «Entities» та вибору вкладки «Individuals».

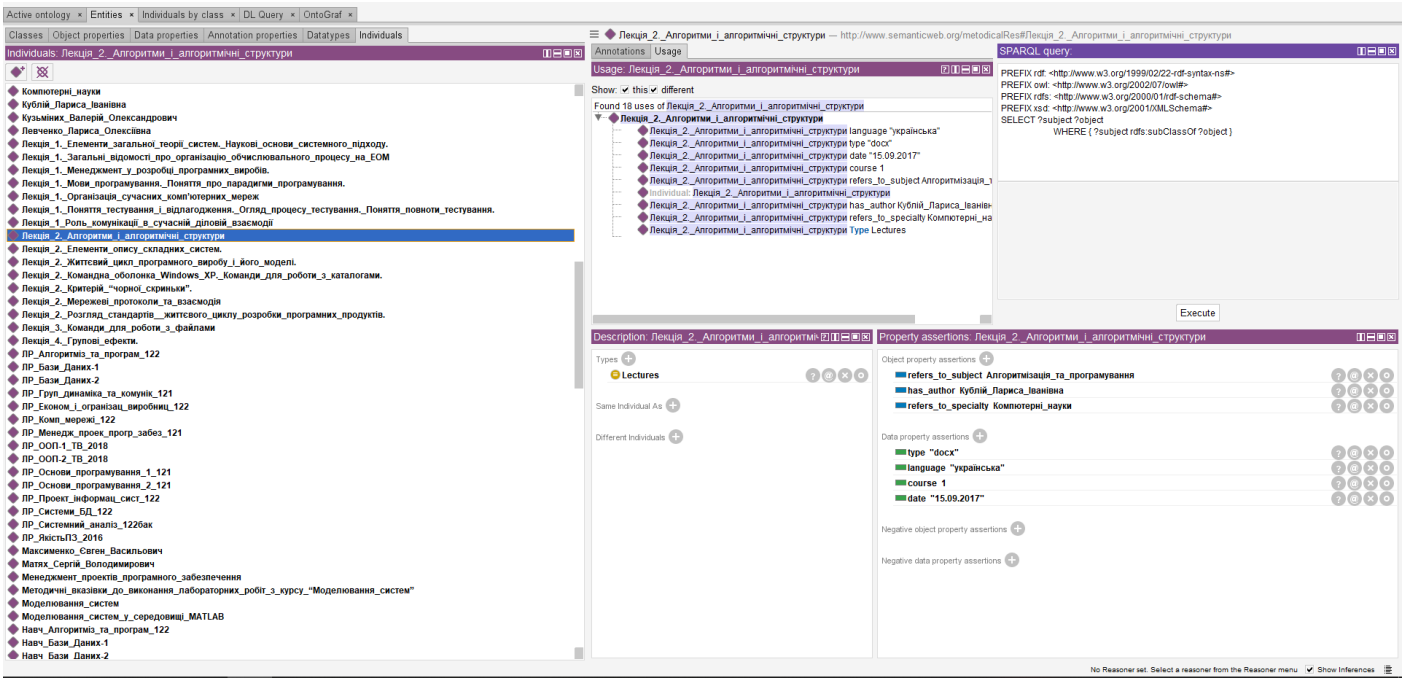


Рисунок 4.12 – Заповнення екземплярів класів даними

Таким чином було створено два онтологічних файли для кожної предметної області. Можна сказати, що заповнення онтологій даними – це досить тривалий процес, але результатом буде досить структурована база даних, працювати з якою зможе звичайний користувач.

4.3. Розробка SPARQL запитів до онтологій

Під час побудови програми, було використано SPARQL запити різних видів: одні виводили назви дисциплін за заданим семестром чи роком навчання, інші ж відображали інформацію про кількість заліків/екзаменів у певному семестрі. Пропонується розглянути найбільш вживані запити.

Перед використанням бібліотеки RDF та фреймворку .Net для побудови інтерфейсу клієнта, спочатку необхідно протестувати на правильність майбутні SPARQL запити. Це можна виконати прямо у програмі Protégé у вікні «SPARQL query» (рисунок 4.14).

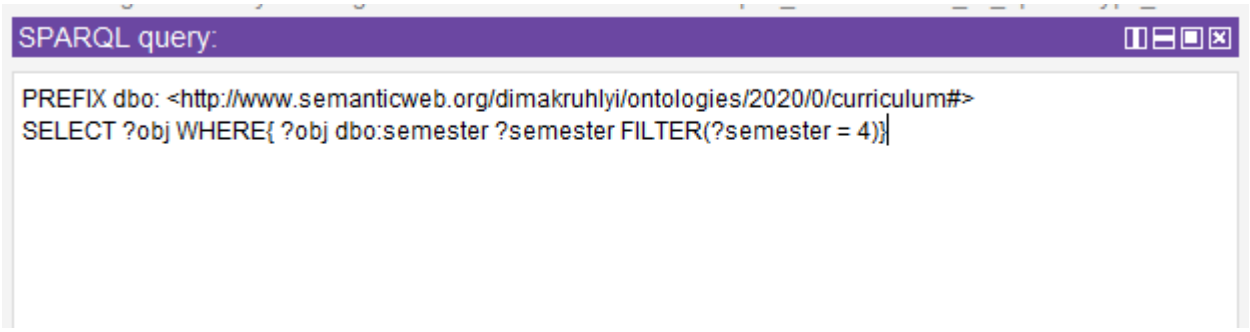


Рисунок 4.13 – Приклад SPARQL запиту для виведення предметів у 4-му семестрі

Результатом поданого вище SPARQL запиту буде виведено навчальні дисципліни студентів кафедри АПЕПС в 4-му семестрі (рисунок 4.15).

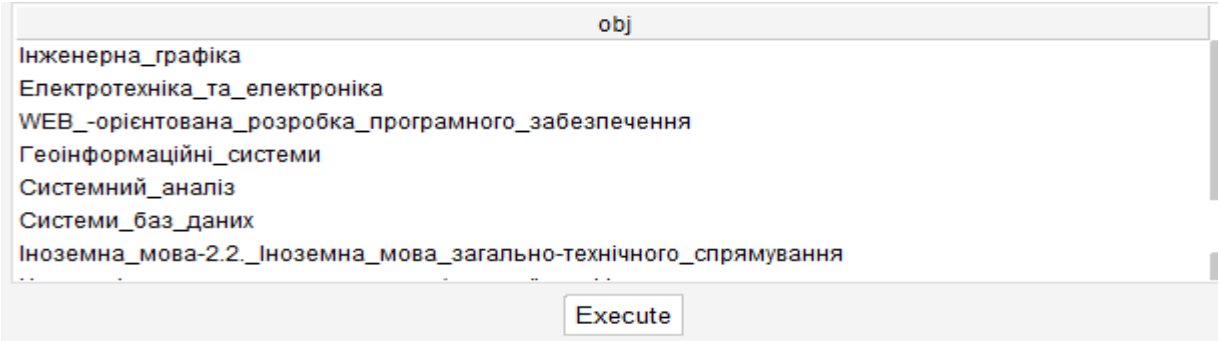


Рисунок 4.14 – Виведення результату SPARQL запиту

Другий тип SPARQL запиту є запит, який виводить всі дисципліни тип здачі який в кінці семестру є екзамен (рисунок 4.16).

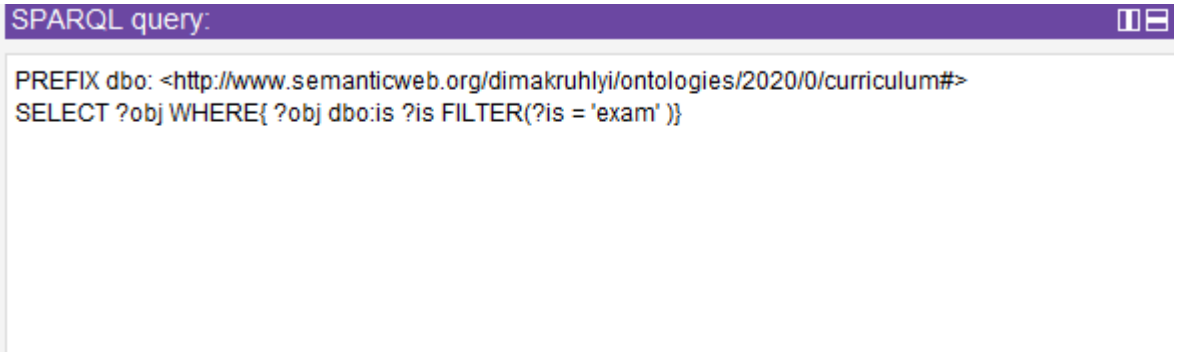


Рисунок 4.15 – Приклад SPARQL запиту для виведення предметів з яких студенти складають екзамен

Результатом даного типу запитів буде список навчальних дисциплін data property яких дорівнює «exam» (рисунок 4.17).

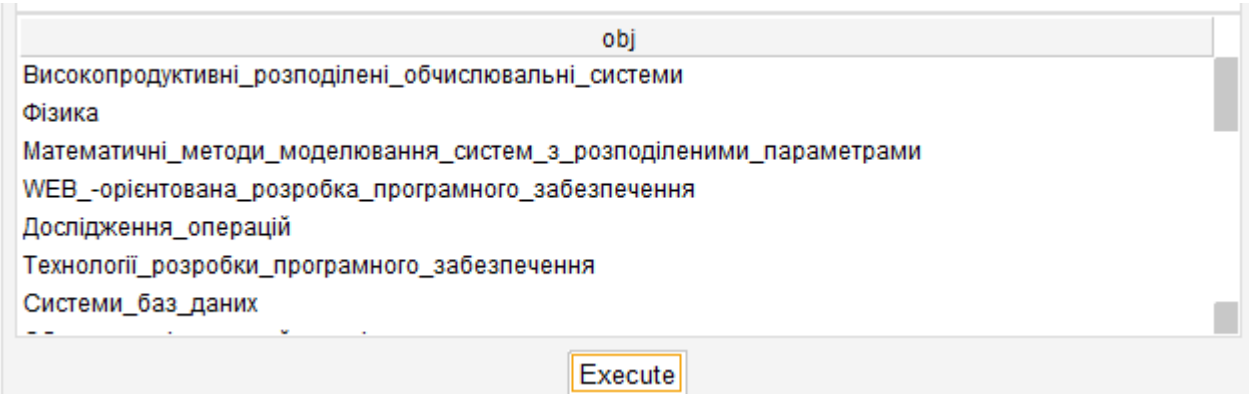


Рисунок 4.16 – Виведення результату SPARQL запиту

Отже, було розглянуто декілька прикладів основних SPARQL запитів, які використовуються у розробці даного програмного продукту. Також спочатку можна

тестувати такі запити безпосередньо в програмі Protégé, а вже після того, впевнившись у правильності запиту, переносити його в код програмного продукту.

4.4. Написання інтерфейсу користувача за онтологічними запитами

Спираючись про наведену вище інформацію про незручність перегляду даних в Protégé, виникає потреба в створенні зручного інтерфейсу користувача, який би дозволяв переглядати дані у зручному вигляді.

Інтерфейс користувача та реалізація SPARQL запитів була написана за допомогою об'єктно орієнтованої мови програмування C# з використанням платформи .Net Framework та бібліотеки RDF. Сам же інтерфейс було реалізовано за допомогою Windows Form. Було додано всі обов'язкові поля, вкладки та кнопки і прив'язані events до них. Event-ом є звичайний обробник подій, який прикладається на кнопки. Тобто при нажатті на кнопку буде відбуватися якась дія. Це дозволяє користувачеві дуже зручно взаємодіяти з інтерфейсом програми та надсилати запити до створених онтологій.

Так як всі інформаційні ресурси знаходяться в Protégé в одному місці, створюваний інтерфейс повинен містити поля, які дають змогу вибирати відповідну категорію для пошуку. Переглядаючи результат виконання SPARQL запиту у редакторі онтологій (рисунок 4.14, рисунок 4.16), ми лише бачимо назви знайдених об'єктів даних. Protégé не дає нам можливості шляхом натискання на відповідний об'єкт переглядати детальну інформацію про нього (властивості об'єктів та властивості даних). Також, якщо це файл методичного або організаційного матеріалу, то ми можемо бачити лише його назву. Створюваний інтерфейс користувача дає можливість, шляхом подвійного натискання лівої кнопки миші по назві, переглянути детальну інформацію про навчальні дисципліни та відкрити знайдений файл методичних матеріалів.

Що ж стосовно самого коду програми, то серед нього можна виявити методи різних типів та з різною функціональністю. Першою ж групою методів, є методи, які безпосередньо взаємодіють з елементами форми, такі як selectCourse_SelectedIndexChanged, message, resultOutput4_CellDoubleClick, find9_Click. Ці методи є обробниками подій і реагують на зміну стану кнопок чи comboBox. Другим типом методів є методи, які вже відправляють до створених онтологій SPARQL запити. До них належать: semesterBtn_Click, typeBtn_Click, cathedraBtn_Click, MetodicalResourcesQuery та інші. Їхньою основною задачею є зчитування даних з файлу, отримання даних від клієнта з форми, відправлення запиту до онтології, порівняння значень з тим, що запитує клієнт та виведення результату клієнту у спеціальне поле Data Grid View.

4.5. Висновки до розділу

В даному розділі було розглянуто інструментальні засоби для реалізації програмного застосунку та побудови онтологічних файлів. Було досліджено різні типу SPARQL запитів та розглянуто основні методи програмного коду.

5. РОБОТА КОРИСТУВАЧА З ПРОГРАМНИМ ПРОДУКТОМ

Так як даний програмний продукт було розроблено в середовищі Visual Studio 2017 за допомогою мови програмування C#, то існують певні вимоги до програмного забезпечення для успішного запуску створеної програмної системи.

5.1. Системні вимоги

Для запуску та використання даної програми на іншому ПК необхідна перш за все наявність на ньому установленого Microsoft .NET Framework 4, оскільки програма була написана з використанням даної платформи. Також ПК повинен мати операційну систему Windows 7 або Windows 10. Крім того, необхідний процесор Intel Core Pentium з частотою мінімум 1.6 GHz. Для відкриття файлів методичних матеріалів необхідна наявність встановленого пакету програм Microsoft Office 2010 або вищої версії.

Найпростішим способом для запуску даної програми являється запуск файлу diplom.exe, що знаходиться за шляхом /diplom/bin/Debug. Для запуску створеної програми іншим способом необхідна наявність установленого середовища Visual Studio 2017. Необхідно відкрити папку з проектом через це середовище та натиснути кнопку «Start». Але якщо ж виникає потреба в редагуванні онтології, то необхідно мати встановлену програму Protégé 5.5.0.

5.2. Робота користувача з системою

Відразу після запуску файлу diplom.exe, користувач бачить робочу область програми (рисунок 5.1).

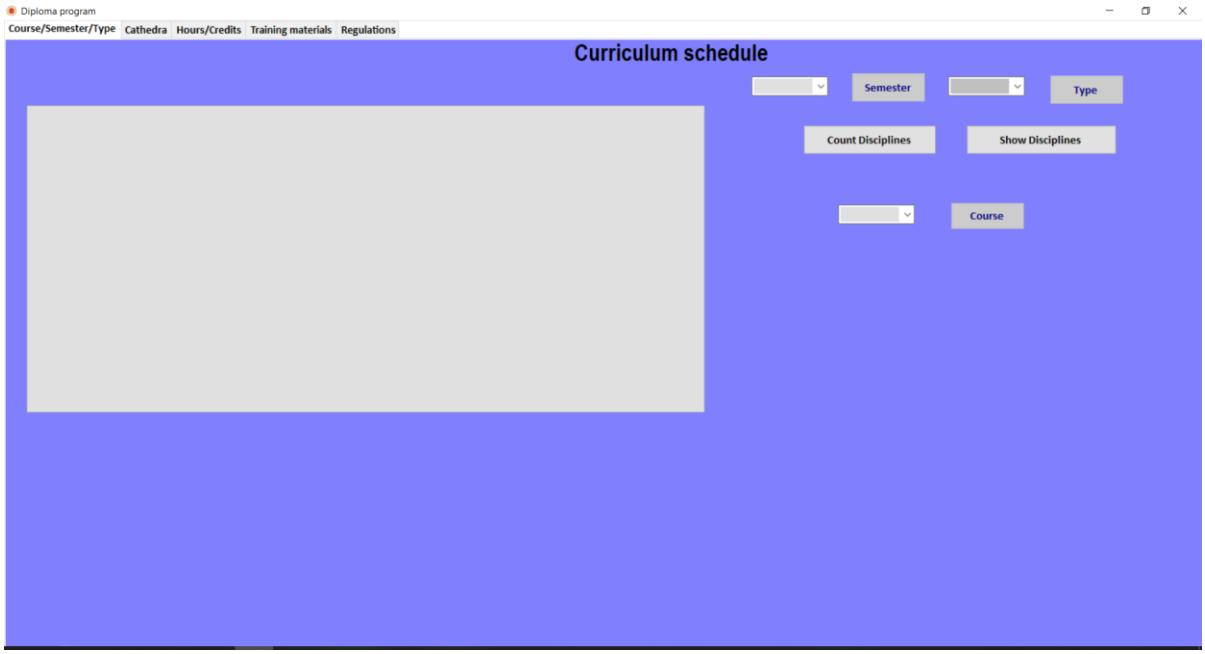


Рисунок 5.1 – Робоча область програми при запуску

Авторизація користувача не відбувається, тому що сфера застосування і коло осіб, які будуть використовувати дану програму, не потребує обмеженого доступу даних для користувача певного типу. Як видно з рисунок 5.1, програма має п’ять вкладок:

- Course/Semester/Type;
- Cathedra;
- Hours/Credits;
- Training materials;
- Regulations.

Тож розглянемо тепер більш детально кожну вкладку.

Перша вкладка дозволяє користувачу переглядати навчальні дисципліни за такими критеріями: «semester» (1,2,3..), «type» (залік, курсова, екзамен), «course» (1,2,3..), а також підраховувати та переглядати, наприклад, «кількість екзаменів у 8-му семестрі». Це доволі зручно, адже це дозволяє зекономити час на пошук необхідної дисципліни викладачем чи студентом. На рисунок 5.2 відображено дисципліни за 4-й курс за спеціалізацією «геометричне моделювання в інформаційних системах».

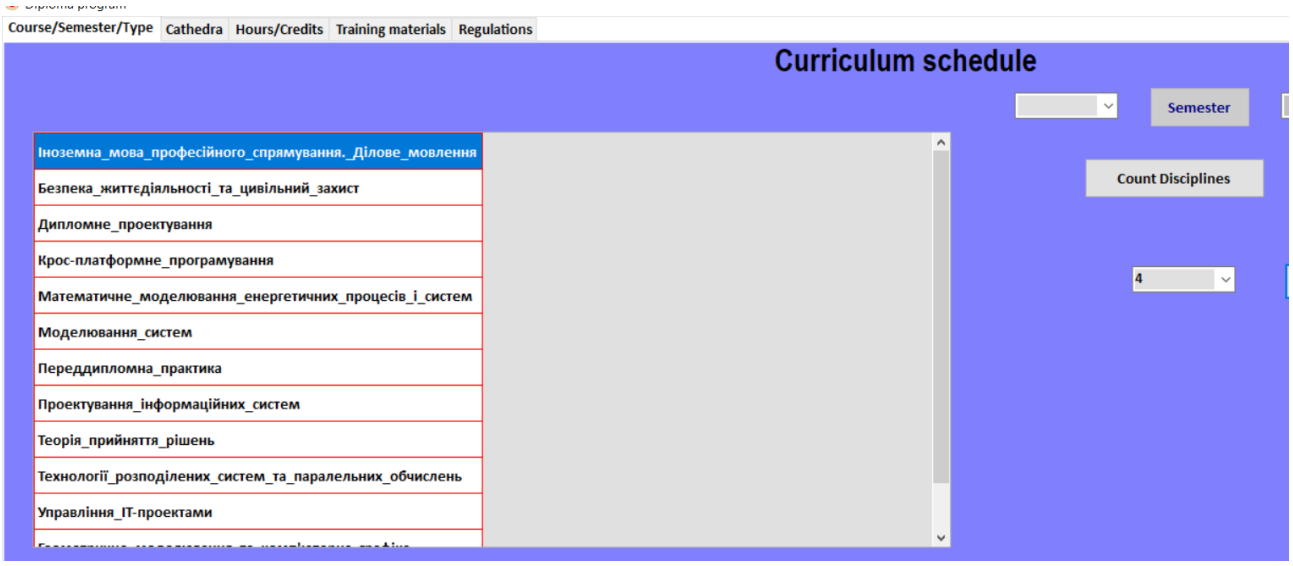


Рисунок 5.2 – Відображення навчальних дисциплін

Подвійний клік мишки по дисципліні дає можливість переглянути більш детальну інформацію про цю дисципліну (рисунок 5.3).

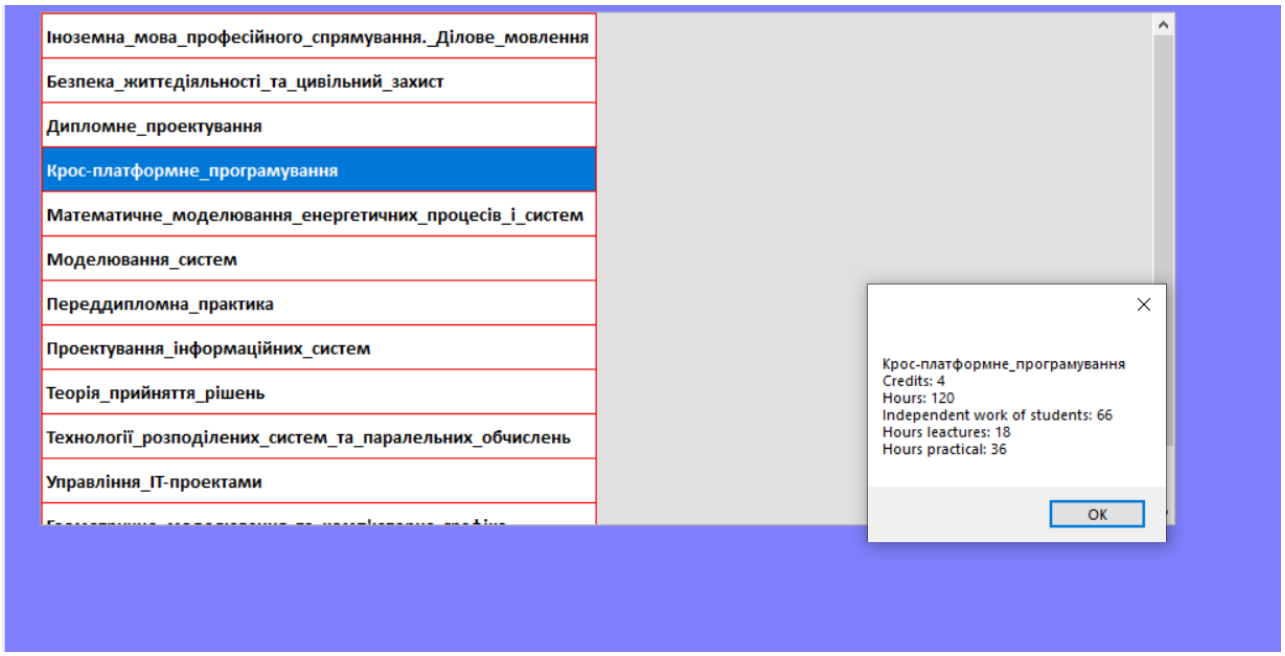


Рисунок 5.3 – Перегляд інформації про Крос-платформне програмування

Також, як було сказано вище, дана вкладка дозволяє переглядати кількість заліків/екзаменів/курсових за певний семестр. Це продемонстровано на рисунок 5.4.

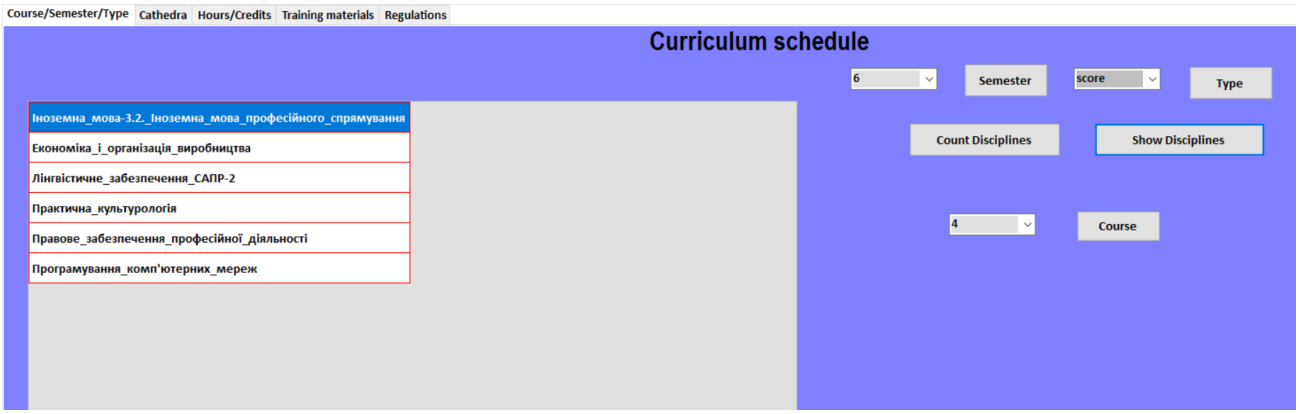


Рисунок 5.4 – Перегляд заліків за 6-й семестр

Якщо введених користувачем в онтологічній базі даних не було знайдено, то виводиться відповідне повідомлення (рисунок 5.5).

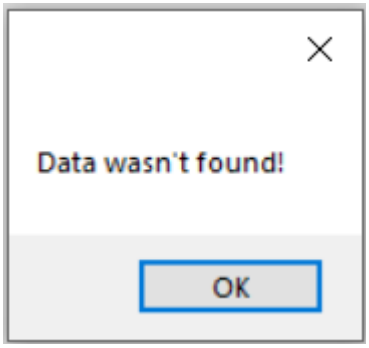


Рисунок 5.5 – Виведення вікна, якщо дані не було знайдено

Друга вкладка даного програмного продукту дозволяє користувачеві переглядати навчальні дисципліни по кафедрах (рисунок 5.6).

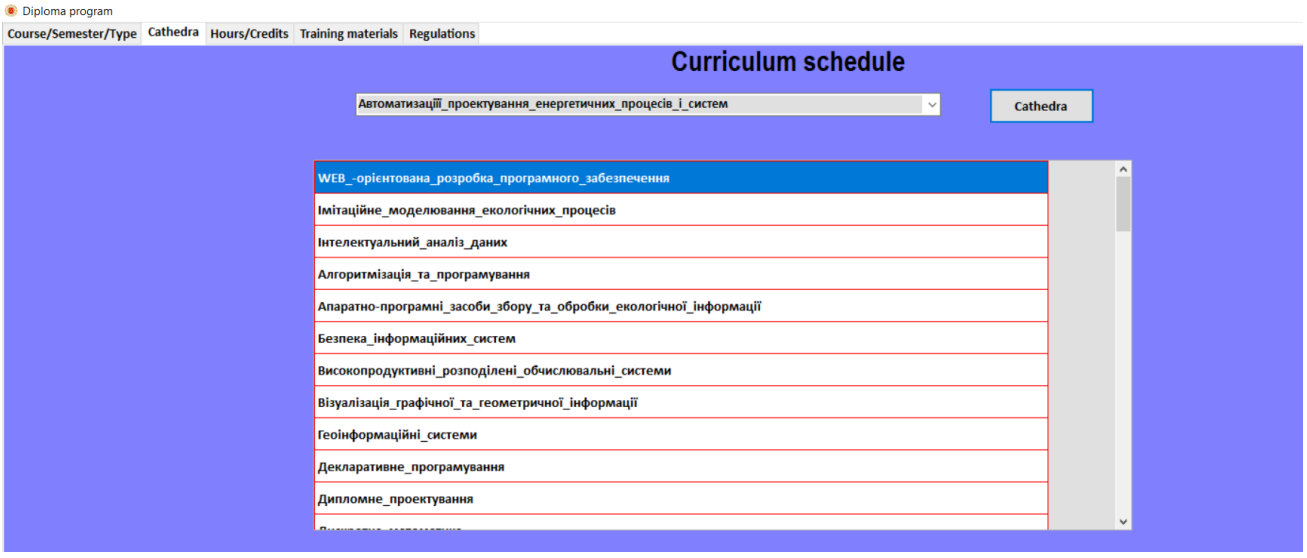


Рисунок 5.6 – Перегляд дисциплін кафедри АПЕПС

Третя вкладка дозволяє відсортувати необхідні дисципліни за критеріями кредитів, кількості лабораторних/лекційних годин та тп. Тут знаходяться три кнопки «Less Than», «More Than» та «Equal», що дозволяють відсортовувати дані в онтології та виводити результат користувачу (рисунок 5.7). Адже не завжди користувач знає точну кількість кредитів чи годин предмету, інформацію про який йому необхідно знайти.

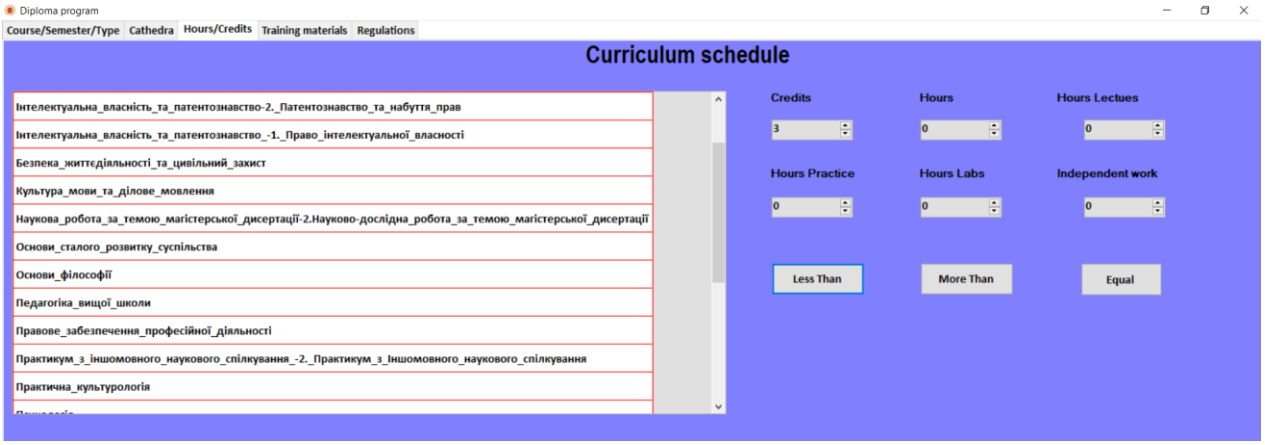


Рисунок 5.7 – Відображення дисциплін, кількість кредитів яких менша за 3

Четверте ж вікно програми дає можливість користувачу переглянути і знайти необхідний методичний матеріал за критеріями спеціальності, предмет, а також викладачем, до якого відноситься шуканий матеріал. Крім того, є можливість переглянути освітні програми, робочі програми та анотації. Нас рисунок 5.8 відображено знайдені методичні матеріали за викладачем «Кублій Лариса Іванівна».

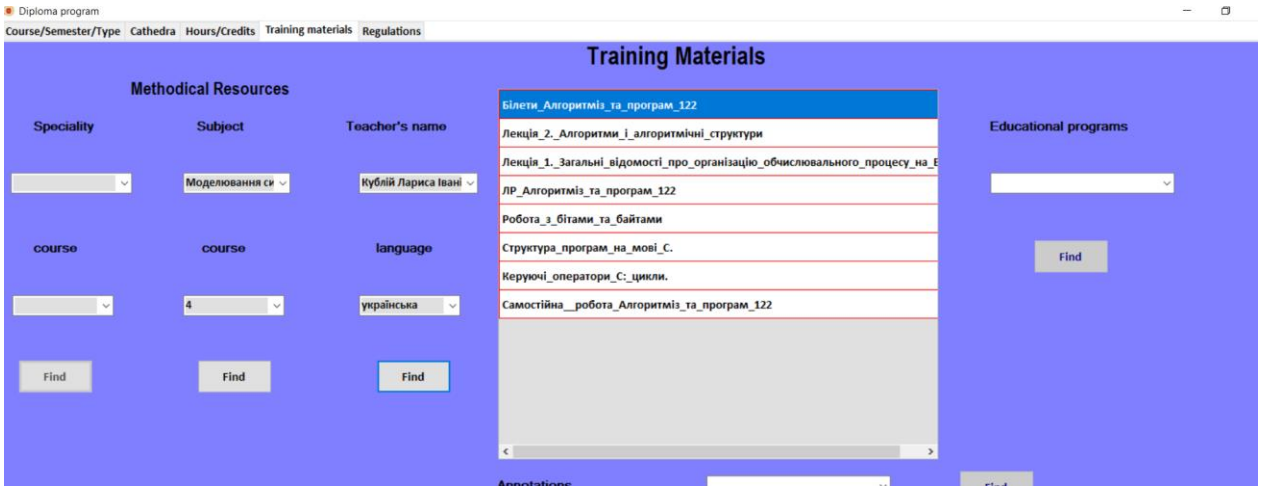


Рисунок 5.8 – Відображення методичних матеріалів за викладачем

У випадку, якщо ж ніяких методичних матеріалів чи освітніх програм не було знайдено, то з’являється відповідне повідомлення (рисунок 5.5).

Подвійний клік на необхідний методичний матеріал дає змогу користувачу відкрити файл засобами Microsoft Office Word (рисунок 5.9).

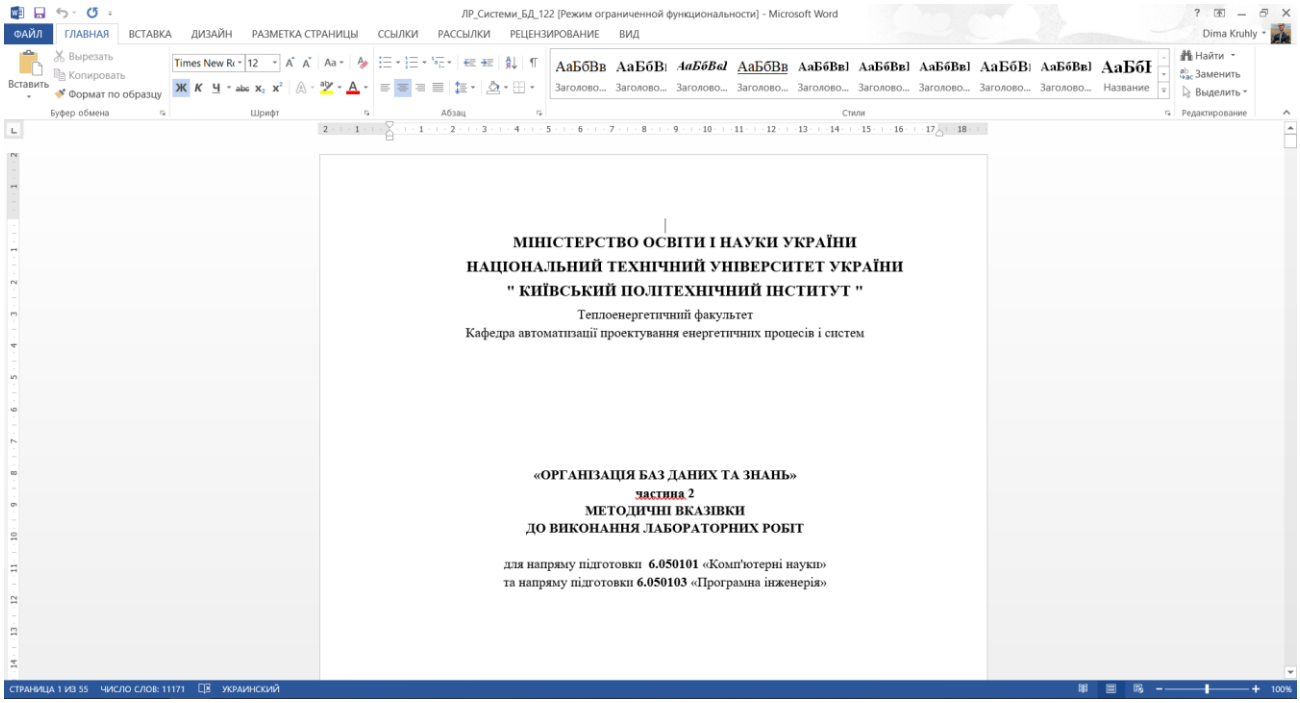


Рисунок 5.9 – Відкриття вибраного методичного файлу

Ця функція дає змогу користувачу відкривати знайдені файли безпосередньо із вікна програми, завдяки чому він (користувач) зразу ж може перевірити достовірність знайденої інформації. У випадку, якщо даний файл не можна відкрити, або він відсутній, то виводиться відповідне повідомлення (рисунок 5.10).

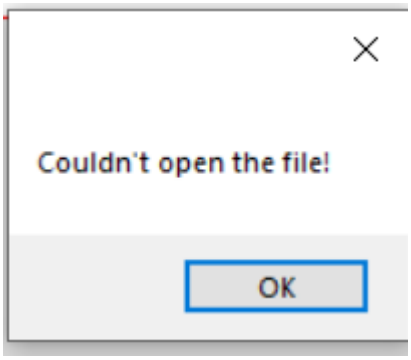


Рисунок 5.10 – Виведення вікна у випадку неможливості відкриття файлу

П’ята вкладка програми дозволяє здійснити пошук за двома категоріями: спеціальністю та курсом. Вона дозволяє шукати нормативні документи та навчальні програми користувачем, а також відкривати їх з вікна програми (рисунок 5.11).

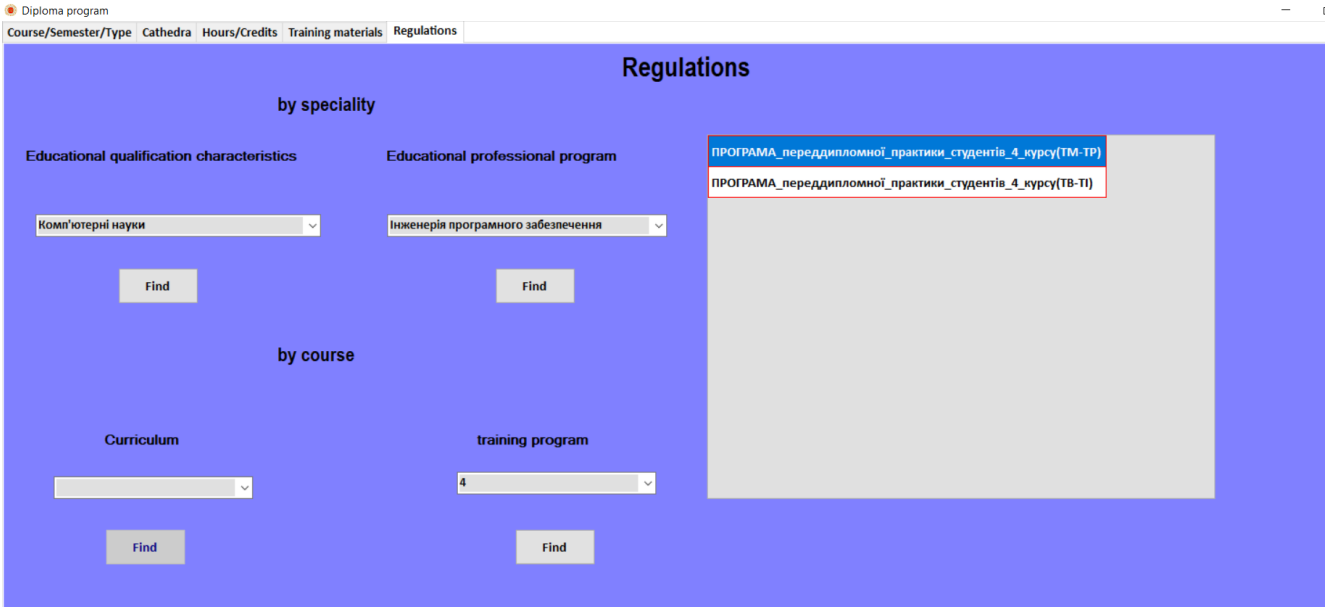


Рисунок 5.11 – Виведення навчальних програм 4-го курсу

У випадку, якщо даний файл не можна відкрити, або він відсутній, то виводиться відповідне повідомлення (рисунок 5.10).

5.3. Висновки до розділу

Підсумовуючи наведену вище інформацію, можна зробити висновок, що було розглянуто основні особливості та функції системи. Було наведено приклади роботи системи та випадки, коли інформацію не було знайдено. А також розглянуто основні вимоги до ПК у разі експлуатування системи на іншому пристрої.

ВИСНОВКИ

В ході виконання дипломної роботи було розроблено формалізацію онтології навчальних дисциплін та методичних ресурсів. Під час роботи над даною задачею було розглянуто предметну область даної системи, що складається з навчальних планів та методичних матеріалів кафедри. Під час вибору програмного інструментарію для створення та редагування онтологій було розглянуто та аналізовано декілька варіантів програм, розглянуто переваги та недоліки кожної, а також аргументовано вибір саме редактора Protégé 5.5.0.

Під час розробки вже самої програмної системи, було побудовано дві онтологічні структуру формату .owl. Перший файл містить дерево класів предметної області навчальних планів та дисциплін, в той час як другий - дерево класів методичних матеріалів та інших інформаційних ресурсів. Після чого було розроблено і протестовано SPARQL запити до кожної онтології за відповідними критеріями. Попередньо протестувавши дані запити в Protégé 5.5.0 та впевнившись в їх правильності, було розроблено клієнтський інтерфейс програми, який дозволяє користувачу здійснювати SPARQL запити до онтологічних файлів та отримувати необхідні дані з них.

Розроблено програмний інструментарій, який дає змогу користувачу, що поняття не має, що таке SPARQL запити та онтології, взаємодіяти з онтологіями шляхом надсилання запитів. Також було розглянуто поняття онтології, інформаційних ресурсів, SPARQL запиту.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні матеріали, які вони бувають [Електронний ресурс] – Режим доступу до ресурсу: https://zadybie-school.ucoz.ru/pedagog/metodicheskie_materialy.htm.

2. Інформаційні ресурси як об'єкт інформаційних правовідносин [Електронний ресурс] // Інформаційне право. – 2018. – Режим доступу до ресурсу: http://pgp-journal.kiev.ua/archive/2018/4/30.pdf.

3. Навчальний план навчального закладу [Електронний ресурс] – Режим доступу до ресурсу: https://zaochnik.com/spravochnik/pedagogika/teorija-obuchenija/uchebnyj-plan/.

4. Освітні програми: Рекомендації до розроблення [Текст] / Уклад. В. П. Головенкін. – К. : КПІ ім. Ігоря Сікорського, 2018. – 39 с.

5. Gruber, T., A translation approach to portable ontologies, Knowledge Acquisition, 5 (1993), 199–220.

6. Gruber T.R. The role of common ontology in achieving sharable, reusable knowledge bases // Principles of Knowledge Representation and Reasoning. Proceedings of the Second International Conference. J.A. Allen, R. Fikes, E. Sandewell – eds. Morgan Kaufmann, 1991, 601-602.

7. Спрощене представлення OWL онтологій для їх застосування в графічних користувацьких інтерфейсах [Електронний ресурс]. – 2012. – Режим доступу до ресурсу: https://cyberleninka.ru/article/n/uproschennoe-predstavlenie-owl-ontologiy-dlya-ih-primeneniya-v-graficheskih-polzovatelskih-interfeysah/viewer.

8. Gruber, T., Toward principles for the design of ontologies used for knowledge sharing, International Journal of Human-Computer Studies, 43 (1965), 907–928.

9. OWL Web Ontology Language Overview [Електронний ресурс] // W3C. – 2004. – Режим доступу до ресурсу: https://www.w3.org/TR/owl-features/.

10. Automatic Integration and Querying of Semantic Rich Heterogeneous Data [Електронний ресурс] // Managing the Web of Things. – 2017. – Режим доступу до ресурсу: https://www.sciencedirect.com/topics/computer-science/resource-description-framework.

11. Frameworks for knowledge representation [Електронний ресурс] // Towards a Semantic Web. – 2011. – Режим доступу до ресурсу: https://www.sciencedirect.com/topics/computer-science/web-ontology-language.

12. RDF Schema RDFS [Електронний ресурс] – Режим доступу до ресурсу: https://www.obitko.com/tutorials/ontologies-semantic-web/rdf-schema-rdfs.html.

13. DAML+OIL [Електронний ресурс] // Department of Computer Science, University of Manchester Oxford Road. – 2002. – Режим доступу до ресурсу: https://www.cs.ox.ac.uk/people/ian.horrocks/Publications/download/2002/ieeede2002.pdf.

14. What is SPARQL? [Електронний ресурс] – Режим доступу до ресурсу: https://www.ontotext.com/knowledgehub/fundamentals/what-is-sparql/.

15. Ontology Development Tools for Ontology-Based Knowledge Management. // Encyclopedia of E-Commerce, E-Government, and Mobile Commerce / , 2006. – С. 859–861.

16. G. Guizzardi, R.A. Falbo, J.G. Pereira Filho, Using Objects and Patterns to Implement Domain Ontologies, Journal of the Brazilian Computer Society, vol.1, n. 8, July 2002.

17. R.A. Falbo, C.S. Menezes, and A.R.C. Rocha, Using Ontologies to Improve Knowledge Integration in Software Engineering Environments, in Proceedings of SCI’98/ISAS’98, Orlando, USA, July, 1998.

18. OntoBroker and OntoStudio X [Електронний ресурс] – Режим доступу до ресурсу: https://www.semafora-systems.com/ontobroker-and-ontostudio-x.

19. Assessment of Ontology Development. // Information Systems Development: Business Systems and Services: Modeling and Development / , 2011. – С. 325–326.

20. Protege Stanford [Електронний ресурс] – Режим доступу до ресурсу: https://protege.stanford.edu/.

21. Visual Studio 2017 Overview And New Features [Електронний ресурс]. – 2017. – Режим доступу до ресурсу: https://www.c-sharpcorner.com/article/visual-studio-2017-overview-and-new-features/.

22. What is .NET Framework? Complete Architecture Tutorial [Електронний ресурс] – Режим доступу до ресурсу: https://www.guru99.com/net-framework.html.

23. dotNetRDF [Електронний ресурс] – Режим доступу до ресурсу: https://github.com/dotnetrdf/dotnetrdf.

ДОДАТОК 1

Онтологічна система інформаційних ресурсів кафедри

Специфікація

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТР61110_20Б

Аркушів 2

Київ – 2020

Позначення	Найменування	Примітки
Документація		
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТР61 110_20Б 81-1	Дипломна записка, Круглий Д., ТР-61.docx	Пояснювальна записка
Компоненти		
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТР61 110_20Б 12-1	curriculum.owl	Онтологічна система даних навчальних дисциплін
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТР61 110_20Б 12-2	documents.owl	Онтологічна система даних методичних матеріалів
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТР61 110_20Б 12-3	Form1.cs	Програмний код клієнтської частини продукту
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТР61 110_20Б 12-4	diplom.sln	Файл запуску проекта
УКР.НТУУ «КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТР61 110_20Б 13-1	опис.docx	Опис програмного коду

ДОДАТОК 2

Онтологічна система інформаційних ресурсів кафедри

Лістинг програми

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТР61110_20Б 12-3

Аркушів 14

Київ – 2020

```
using System;// Підключення бібліотек
using System.Web;
using System.Windows.Forms;
using VDS.RDF;
using VDS.RDF.Query;
// Оголошення namespace
namespace diplom
{
   public partial class Curriculum_schedule : Form
   {
      //Додаткові поля
      string newCombText;
      string query;
      //Конструктор
      public Curriculum_schedule(){
         InitializeComponent();
      }
      //Отримання назв без префіксів
      private string Split(string a){
         string[] arr = a.Split('#');
         return arr[1].ToString();
      }
      private void semesterBtn_Click(object sender, EventArgs e){
         // Зчитування онтологічних даних з файлу
         string result = null;
         IGraph graph = new Graph();
         graph.LoadFromFile("curriculum.owl");
         int semester = Convert.ToInt32(selectSemestr.SelectedItem); // Беремо
значення з форми
         Object output = graph.ExecuteQuery("PREFIX dbo:
<http://www.semanticweb.org/dimakruhlyi/ontologies/2020/0/curriculum#>" + "SELECT
?obj WHERE{ ?obj dbo:semester ?semester FILTER(?semester =" + semester + ")}"); //
Створення SPARQL запиту
         if (output is SparqlResultSet){
            resultOutput.Rows.Clear(); // Очищуємо поле виводу даних
            SparqlResultSet resultSet = (SparqlResultSet)output;
            foreach (SparqlResult resultItem in resultSet){
               result = Split(System.Web.HttpUtility.UrlDecode(resultItem.ToString()));
// Виведення результату
               resultOutput.Rows.Add(result);
            }
```

```
            }
        }
        private void courseBtn_Click(object sender, EventArgs e){
            // Зчитування онтологічних даних з файлу
            string result = null;
            IGraph graph = new Graph();
            graph.LoadFromFile("curriculum.owl");
            int course = Convert.ToInt32(selectCourse.SelectedItem);// Беремо значення з
форми
            object output = graph.ExecuteQuery("PREFIX dbo:
<http://www.semanticweb.org/dimakruhlyi/ontologies/2020/0/curriculum#>" + "SELECT
?obj WHERE{" + "?obj dbo:Course ?Course FILTER(?Course = " + course + ")}"); //
Створення SPARQL запиту
            if (output is SparqlResultSet){
                resultOutput.Rows.Clear();// Очищуємо поле виводу даних
                SparqlResultSet resultSet = (SparqlResultSet)output;
                foreach (SparqlResult resultItem in resultSet){
                    result = Split(System.Web.HttpUtility.UrlDecode(resultItem.ToString()));
// Виведення результату
                    resultOutput.Rows.Add(result);
                }
            }
        }
        private void typeBtn_Click(object sender, EventArgs e){
            // Зчитування онтологічних даних з файлу
            string result = null;
            IGraph graph = new Graph();
            graph.LoadFromFile("curriculum.owl");
            string type = selectType.Text;// Беремо значення з форми
            Object output = graph.ExecuteQuery("PREFIX dbo:
<http://www.semanticweb.org/dimakruhlyi/ontologies/2020/0/curriculum#>" + "SELECT
?obj WHERE{ ?obj dbo:is ?is FILTER(?is ='" + type + "')}"); // Створення SPARQL запиту
            if (output is SparqlResultSet){
                resultOutput.Rows.Clear(); // Очищуємо поле виводу даних
                SparqlResultSet resultSet = (SparqlResultSet)output;
                foreach (SparqlResult resultItem in resultSet){
                    result = Split(System.Web.HttpUtility.UrlDecode(resultItem.ToString()));
// Виведення результату
                    resultOutput.Rows.Add(result);
                }
            }
```

```
        }
        private void countBtn_Click(object sender, EventArgs e){
            // Зчитування онтологічних даних з файлу
            string result = null;
            IGraph graph = new Graph();
            graph.LoadFromFile("curriculum.owl");
            int semester = Convert.ToInt32(selectSemestr.SelectedItem);// Беремо
значення з форми
            Object output = graph.ExecuteQuery("PREFIX dbo:
<http://www.semanticweb.org/dimakruhlyi/ontologies/2020/0/curriculum#>" + "SELECT
(COUNT(?semester) AS ?exam)" + "WHERE{ ?obj dbo:is ?is FILTER(?is ='" +
selectType.Text + "')." + "?obj  dbo:semester ?semester FILTER(?semester = " + semester +
")}"); // Створення SPARQL запиту
            if (output is SparqlResultSet){
                resultOutput.Rows.Clear(); // Очищуємо поле виводу даних
                SparqlResultSet resultSet = (SparqlResultSet)output;
                foreach (SparqlResult resultItem in resultSet){
                    result= selectType.Text + "=" +
HttpUtility.UrlDecode(resultItem.ToString()).Split('=')[1].Split('^')[0] + "\r\n"; // Виведення
результату
                    resultOutput.Rows.Add(result);
                }
            }
        }
        private void showDisciplinesBtn_Click(object sender, EventArgs e){
            // Зчитування онтологічних даних з файлу
            string checkResult = null; // Створення змінної для перевірки на пустоту
            IGraph graph = new Graph();
            graph.LoadFromFile("curriculum.owl");
            int semester = Convert.ToInt32(selectSemestr.SelectedItem);// Беремо
значення з форми
            Object output = graph.ExecuteQuery("PREFIX dbo:
<http://www.semanticweb.org/dimakruhlyi/ontologies/2020/0/curriculum#>" + "SELECT
?obj WHERE{ ?obj dbo:is ?is FILTER(?is ='" + selectType.Text + "')" + "?obj dbo:semester
?semester FILTER(?semester = " + semester + ")}"); // Створення SPARQL запиту
            if (output is SparqlResultSet){
                resultOutput.Rows.Clear(); // Очищуємо поле виводу даних
                SparqlResultSet resultSet = (SparqlResultSet)output;
                foreach (SparqlResult resultItem in resultSet){
                    checkResult = HttpUtility.UrlDecode(resultItem.ToString()).Split('#')[1]; //
Виведення результату
```

```
                    resultOutput.Rows.Add(checkResult);
                }
                // Перевірка на пустоту, якщо даних немає, то виведення повідомлення
                if (checkResult == null){
                    MessageBox.Show("Data wasn't found!");
                }
            }
        }
        private void cathedraBtn_Click(object sender, EventArgs e){
            // Зчитування онтологічних даних з файлу
            string result = null;
            IGraph graph = new Graph();
            graph.LoadFromFile("curriculum.owl");
            string cathedra = selectCathedra.Text;// Беремо значення з форми
            Object output = graph.ExecuteQuery("PREFIX dbo:
<http://www.semanticweb.org/dimakruhlyi/ontologies/2020/0/curriculum#>" + "SELECT
?obj WHERE{ ?obj dbo:is_taught dbo:" + cathedra + "}"); // Створення SPARQL запиту
            if (output is SparqlResultSet){
                resultOutput2.Rows.Clear(); // Очищуємо поле виводу даних
                SparqlResultSet resultSet = (SparqlResultSet)output;
                foreach (SparqlResult resultItem in resultSet){
                    //resultOutput20.Text +=
HttpUtility.UrlDecode(resultItem.ToString()).Split('#')[1] + "\r\n"; // Виведення результату;
                    result = Split(System.Web.HttpUtility.UrlDecode(resultItem.ToString()));
                    resultOutput2.Rows.Add(result);
                }
            }
        }
        private void moreBtn_Click(object sender, EventArgs e){
            // Зчитування онтологічних даних з файлу
            string result = null;
            IGraph graph = new Graph();
            graph.LoadFromFile("curriculum.owl");
            double credits = Convert.ToDouble(Math.Round(numCredits.Value, 0));//
Беремо значення з форми
            if (credits != 0){
                Object output = graph.ExecuteQuery("PREFIX dbo:
<http://www.semanticweb.org/dimakruhlyi/ontologies/2020/0/curriculum#>" + "SELECT
?obj WHERE{ ?obj dbo:credits ?credits FILTER(?credits > " + credits + ")}"); // Створення
SPARQL запиту
                if (output is SparqlResultSet){
```

```
                    resultOutput3.Rows.Clear(); // Очищуємо поле виводу даних
                    SparqlResultSet resultSet = (SparqlResultSet)output;
                    foreach (SparqlResult resultItem in resultSet){
                        result =
Split(System.Web.HttpUtility.UrlDecode(resultItem.ToString()));
                        resultOutput3.Rows.Add(result); // Виведення результату;
                    }
                }
            }
            int hours = Convert.ToInt32(Math.Round(numHours.Value, 0));// Беремо
значення з форми
            if (hours != 0){
                Object output = graph.ExecuteQuery("PREFIX dbo:
<http://www.semanticweb.org/dimakruhlyi/ontologies/2020/0/curriculum#>" + "SELECT
?obj WHERE{ ?obj dbo:hours ?hours FILTER(?hours > " + hours + ")}"); // Створення
SPARQL запиту
                if (output is SparqlResultSet){
                    resultOutput3.Rows.Clear(); // Очищуємо поле виводу даних
                    SparqlResultSet resultSet = (SparqlResultSet)output;
                    foreach (SparqlResult resultItem in resultSet){
                        result =
Split(System.Web.HttpUtility.UrlDecode(resultItem.ToString()));
                        resultOutput3.Rows.Add(result); // Виведення результату;
                    }
                }
            }
            int lectures = Convert.ToInt32(Math.Round(numHoursLectures.Value, 0));//
Беремо значення з форми
            if (lectures != 0){
                Object output = graph.ExecuteQuery("PREFIX dbo:
<http://www.semanticweb.org/dimakruhlyi/ontologies/2020/0/curriculum#>" + "SELECT
?obj WHERE{ ?obj dbo:hours_lectures ?hours_lectures FILTER(?hours_lectures > " +
lectures + ")}"); // Створення SPARQL запиту
                if (output is SparqlResultSet){
                    resultOutput3.Rows.Clear(); // Очищуємо поле виводу даних
                    SparqlResultSet resultSet = (SparqlResultSet)output;
                    foreach (SparqlResult resultItem in resultSet){
                        result =
Split(System.Web.HttpUtility.UrlDecode(resultItem.ToString()));
                        resultOutput3.Rows.Add(result); // Виведення результату;
                    }
```

```
                }
            }
            int practice = Convert.ToInt32(Math.Round(numHoursPractice.Value, 0));// Беремо значення з форми
            if (practice != 0){
                Object output = graph.ExecuteQuery("PREFIX dbo: <http://www.semanticweb.org/dimakruhlyi/ontologies/2020/0/curriculum#>" + "SELECT ?obj WHERE{ ?obj dbo:hours_practical ?hours_practical FILTER(?hours_practical > " + practice + ")}"); // Створення SPARQL запиту
                if (output is SparqlResultSet){
                    resultOutput3.Rows.Clear(); // Очищуємо поле виводу даних
                    SparqlResultSet resultSet = (SparqlResultSet)output;
                    foreach (SparqlResult resultItem in resultSet){
                        resultOutput3.Text += HttpUtility.UrlDecode(resultItem.ToString()).Split('#')[1] + "\r\n"; // Виведення результату;
                    }
                }
            }
            int labs = Convert.ToInt32(Math.Round(numHoursLabs.Value, 0));// Беремо значення з форми
            if (labs != 0){
                Object output = graph.ExecuteQuery("PREFIX dbo: <http://www.semanticweb.org/dimakruhlyi/ontologies/2020/0/curriculum#>" + "SELECT ?obj WHERE{ ?obj dbo:hours_laboratory ?hours_laboratory FILTER(?hours_laboratory > " + labs + ")}"); // Створення SPARQL запиту
                if (output is SparqlResultSet){
                    resultOutput3.Rows.Clear(); // Очищуємо поле виводу даних
                    SparqlResultSet resultSet = (SparqlResultSet)output;
                    foreach (SparqlResult resultItem in resultSet){
                        resultOutput3.Text += HttpUtility.UrlDecode(resultItem.ToString()).Split('#')[1] + "\r\n"; // Виведення результату;
                    }
                }
            }
            int student = Convert.ToInt32(Math.Round(numIndependentWork.Value, 0));// Беремо значення з форми
            if (student != 0){
                Object output = graph.ExecuteQuery("PREFIX dbo: <http://www.semanticweb.org/dimakruhlyi/ontologies/2020/0/curriculum#>" + "SELECT ?obj WHERE{ ?obj dbo:independent_work_of_students ?independent_work_of_students
```

```
FILTER(?independent_work_of_students > " + student + ")}"); // Створення SPARQL
запиту
                    if (output is SparqlResultSet){
                        resultOutput3.Rows.Clear(); // Очищуємо поле виводу даних
                        SparqlResultSet resultSet = (SparqlResultSet)output;
                        foreach (SparqlResult resultItem in resultSet){
                            result =
Split(System.Web.HttpUtility.UrlDecode(resultItem.ToString()));
                            resultOutput3.Rows.Add(result); // Виведення результату;
                        }
                    }
                }
            }
            // Метод для SPARQL запиту за методичними ресурсами
            private void MetodicalResourcesQuery(string s){
                string result = null;
                IGraph graph = new Graph();
                graph.LoadFromFile("documents.owl");
                try{
                    Object results = graph.ExecuteQuery(s);
                    if (results is SparqlResultSet){
                        SparqlResultSet resultSet = (SparqlResultSet)results;
                        foreach (SparqlResult resultItem in resultSet){
                            result =
Split(System.Web.HttpUtility.UrlDecode(resultItem.ToString()));
                            resultOutput4.Rows.Add(result);
                        }
                        if (result == null){
                            MessageBox.Show("Data wasn't found!");
                        }
                    }
                    else{
                        MessageBox.Show("Data wasn't found!");
                    }
                }
                catch (RdfQueryException queryEx){
                    MessageBox.Show(queryEx.Message);
                }

            }
```

```
            // Пошук за спеціальністю
            private void find1_Click(object sender, EventArgs e){
                resultOutput4.Rows.Clear();
                query = "PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#> PREFIX
metodicalres:<http://www.semanticweb.org/metodicalRes#> SELECT ?o WHERE { ?o
rdf:type ?type. ?type rdfs:subClassOf* metodicalres:Educational_materials. ?o
metodicalres:refers_to_specialty metodicalres:" + RenameSpeciality(specialityComboBox) +
". ?o  metodicalres:course ?course. filter(?course = " + course1ComboBox.Text + ")} ";
                MetodicalResourcesQuery(query);
            }
            //Пошук за предметом
            private void find3_Click(object sender, EventArgs e){
                resultOutput4.Rows.Clear();
                query = "PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#> PREFIX
metodicalres:<http://www.semanticweb.org/metodicalRes#> SELECT ?o WHERE { ?o
rdf:type ?type. ?type rdfs:subClassOf* metodicalres:Educational_materials. ?o
metodicalres:refers_to_subject  metodicalres:" + RenameSubject(subjectComboBox) + ". ?o
metodicalres:course ?course. filter(?course = " + course2comboBox.Text + ")} ";
                MetodicalResourcesQuery(query);
            }
            //Пошук за викладачем
            private void find2_Click(object sender, EventArgs e){
                resultOutput4.Rows.Clear();
                newCombText = teachersComboBox.Text.Replace(' ', '_');
                query = "PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#> PREFIX
metodicalres:<http://www.semanticweb.org/metodicalRes#> SELECT ?o WHERE { ?o
rdf:type ?type. ?type rdfs:subClassOf* metodicalres:Educational_materials. ?o
metodicalres:has_author  metodicalres:" + newCombText + ". ?o metodicalres:language
?language. FILTER regex(?language, '" + languageComboBox.Text + "')}";
                MetodicalResourcesQuery(query);
            }
            // Метод для SPARQL запиту за предметом
            private void DataBySubject(string clas, string subject){
                string result = null;
                IGraph graph = new Graph();
                graph.LoadFromFile("documents.owl");
                try{
```

```
                Object results = graph.ExecuteQuery("PREFIX
rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>  PREFIX
rdfs:<http://www.w3.org/2000/01/rdf-schema#> PREFIX
metodicalres:<http://www.semanticweb.org/metodicalRes#> SELECT ?o WHERE { ?o
rdf:type ?type.?type rdfs:subClassOf* metodicalres:" + clas + ". ?o
metodicalres:refers_to_subject metodicalres:" + subject + ".}");
                if (results is SparqlResultSet)
                {
                   SparqlResultSet resultSet = (SparqlResultSet)results;
                   foreach (SparqlResult resultItem in resultSet){
                      result =
Split(System.Web.HttpUtility.UrlDecode(resultItem.ToString()));
                      resultOutput4.Rows.Add(result);
                   }
                   if (result == null){
                      MessageBox.Show("Data wasn't found!");
                   }
                }
                else{
                   MessageBox.Show("Data wasn't found!");
                }
             }
             catch (RdfQueryException queryEx){
                MessageBox.Show(queryEx.Message);
             }
          }
          //Пошук навчальних програм
          private void find6_Click(object sender, EventArgs e){
             resultOutput4.Rows.Clear();
             DataBySubject("Training", RenameSubject(educationalProgramsComboBox));
          }
          //Пошук робочих програм
          private void find5_Click(object sender, EventArgs e){
             resultOutput4.Rows.Clear();
             DataBySubject("Work_program",
RenameSubject(workingProgramsComboBox));
          }
          //Пошук анотацій
          private void find4_Click(object sender, EventArgs e){
             resultOutput4.Rows.Clear();
             DataBySubject("Annotation", RenameSubject(annotationsComboBox));
```

```
        }

        // Метод для SPARQL запиту за освітніми кваліфікаційними
характеристиками та освітніми професійними програмами
        private void RegulationsBySpeciality(string clas, string spec){
            string result = null;
            IGraph graph = new Graph();
            graph.LoadFromFile("documents.owl");
            try{
                Object results = graph.ExecuteQuery("PREFIX
rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>  PREFIX
rdfs:<http://www.w3.org/2000/01/rdf-schema#> PREFIX
metodicalres:<http://www.semanticweb.org/metodicalRes#> SELECT ?o WHERE { ?o
rdf:type ?type.?type rdfs:subClassOf* metodicalres:" + clas + ". ?o
metodicalres:refers_to_specialty metodicalres:" + spec + ".}");
                if (results is SparqlResultSet){
                    SparqlResultSet resultSet = (SparqlResultSet)results;
                    foreach (SparqlResult resultItem in resultSet){
                        result =
Split(System.Web.HttpUtility.UrlDecode(resultItem.ToString()));
                        resultOutput5.Rows.Add(result);
                    }
                    if (result == null){
                        MessageBox.Show("Data wasn't found!");
                    }
                }
                else{
                    MessageBox.Show("Data wasn't found!");
                }
            }
            catch (RdfQueryException queryEx){
                MessageBox.Show(queryEx.Message);
            }
        }
        //Пошук освітніх професійних програм
        private void find9_Click(object sender, EventArgs e){
            resultOutput5.Rows.Clear();
            RegulationsBySpeciality("Educational_and_professional_programs",
RenameSpeciality(eppComboBox));
        }
```

```
            //Пошук освітніх кваліфікаційних характеристик
            private void find7_Click(object sender, EventArgs e){
                resultOutput5.Rows.Clear();
                RegulationsBySpeciality("Educational_qualification_characteristics",
RenameSpeciality(eqcComboBox));
            }
            // Метод для SPARQL запиту за навчальними планами та тренувальними
програмами за курсом
            private void TrainingProgramCurriculum(string clas, string course){
                string result = null;
                IGraph graph = new Graph();
                graph.LoadFromFile("documents.owl");
                try{
                    Object results = graph.ExecuteQuery("PREFIX
rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>  PREFIX
rdfs:<http://www.w3.org/2000/01/rdf-schema#> PREFIX
metodicalres:<http://www.semanticweb.org/metodicalRes#> SELECT ?o WHERE { ?o
rdf:type ?type.  ?type rdfs:subClassOf* metodicalres:" + clas + ". ?o  metodicalres:course
?course. filter(?course = " + course + ")}");
                    if (results is SparqlResultSet){
                        SparqlResultSet resultSet = (SparqlResultSet)results;
                        foreach (SparqlResult resultItem in resultSet){
                            result =
Split(System.Web.HttpUtility.UrlDecode(resultItem.ToString()));
                            resultOutput5.Rows.Add(result);
                        }
                        if (result == null){
                            MessageBox.Show("Data wasn't found!");
                        }
                    }
                    else{
                        MessageBox.Show("Data wasn't found!");
                    }
                }
                catch (RdfQueryException queryEx){
                    MessageBox.Show(queryEx.Message);
                }
            }

            //Пошук за навчальними програмами
            private void find10_Click(object sender, EventArgs e){
```

```
            resultOutput5.Rows.Clear();
            TrainingProgramCurriculum("Training_programs",
regulationsTrainingProgramComboBox.Text);
        }
        //Пошук за навчальними планами
        private void find8_Click(object sender, EventArgs e){
            resultOutput5.Rows.Clear();
            TrainingProgramCurriculum("Curriculum",
regulationsCurriculumComboBox.Text);
        }
        //Відкриттях файлів по яких клікнули
        private void OpenFile(string name){
            try{
                string commandText = @"C:\Users\Dima Kruhlyi\Desktop\testdocuments\"
+ name + ".doc";
                var open = new System.Diagnostics.Process();
                open.StartInfo.FileName = commandText;
                open.StartInfo.UseShellExecute = true;
                open.Start();
            }
            catch (Exception exception){
                try{
                    string commandText = @"C:\Users\Dima
Kruhlyi\Desktop\testdocuments\" + name + ".docx";
                    var open = new System.Diagnostics.Process();
                    open.StartInfo.FileName = commandText;
                    open.StartInfo.UseShellExecute = true;
                    open.Start();
                }
                catch (Exception except){
                    MessageBox.Show("Couldn't open the file!");
                }
            }
        }

        //Відкриття файлу resultoutput4
        private void resultOutput4_CellDoubleClick(object sender,
DataGridViewCellEventArgs e){
            OpenFile(resultOutput4.CurrentCell.Value.ToString());
        }
        //Відкриття файлу resultoutput5
```

```
            private void resultOutput5_CellDoubleClick(object sender,
DataGridViewCellEventArgs e){
                OpenFile(resultOutput5.CurrentCell.Value.ToString());
            }
            private void message(object sender, EventArgs e){
                string result = resultOutput.CurrentCell.Value.ToString() + "\nCredits: 4
\nHours: 120 \nIndependent work of students: 66 \nHours leactures: 18 \nHours practical:
36";
                MessageBox.Show(result);
            }
            private void message2(object sender, DataGridViewCellEventArgs e){
                MessageBox.Show(resultOutput2.CurrentCell.Value.ToString());
            }
            private void message3(object sender, DataGridViewCellEventArgs e){
                MessageBox.Show(resultOutput3.CurrentCell.Value.ToString());
            }
        }
    }
```

ДОДАТОК 3

Онтологічна система інформаційних ресурсів кафедри

Опис програмного модулю

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТР61110_20Б 13-1

Аркушів 8

Київ – 2020

АНОТАЦІЯ

У даному додатку міститься інформація, що описує програмний продукт для онтологічної системи інформаційних ресурсів кафедри. Даний програмний продукт дозволяє користувачеві, навіть який цілком не знайомий з програмуванням та онтологіями, отримувати дані про освітній процес кафедри шляхом надсилання SPARQL запитів до розроблених онтологій.

Основною перевагою даної програми є те, що інформація міститься не в звичайних SQL таблицях, а в онтології, що забезпечує швидку обробку великих об’ємів даних.

Програма була розроблена у середовищі Visual Studio 2017 мовою програмування C# з використанням .Net Framework та бібліотеки RDF.

ЗМІСТ

ЗАГАЛЬНІ ВІДОМОСТІ

Даний програмний продукт було розроблено в середовищі Visual Studio 2017 мовою програмування C# з використанням .Net Framework та бібліотеки RDF.

Програма призначена як для викладачів, так і для студентів, яким необхідно швидко знайти потрібну інформацію або документ про навчальний процес кафедри. Принцип роботи програми побудовано на відправленні користувачем SPARQL запитів до відповідних онтологій, в яких в свою чергу міститься інформація про навчальний процес.

До основних переваг програми можна віднести швидкість роботи та відносно простий для користувача інтерфейс.

ФУНКЦІОНАЛЬНЕ ПРИЗНАЧЕННЯ

При створенні програмного продукту була поставлена задача зберігання та відображення користувачем великої кількості даних. Так як звичайні бази даних, якщо вони не були попередньо продумані для зберігання великих обсягів інформації, через декілька років втрачають свою ефективність. Тому і було прийнято рішення використати онтологічних підхід для зберігання інформації.

Програмний продукт містить декілька різних типів методів, які виконують наступні функції:

- надсилання SPARQL запитів до онтологічних файлів;
- виведення отриманої інформації користувачем на екран;
- методи event для взаємодії з користувачем (при кліку по кнопці викликається певний метод);
- допоміжні методи, які не взаємодіють з онтологіями, проте покращують взаємодію користувача з програмою.

ОПИС ЛОГІЧНОЇ СТРУКТУРИ

Створена програма складається з декількох частин. Першою і основною частиною ж є онтологічні файли curriculum.owl та documents.owl. До інтерфейсу користувача входить windows form з усіма її компонентами (dataGridView, buttons та ін.).

Робота програми побудована таким чином, що запустивши файл diplom.exe користувач відкриває форму з якою він може взаємодіяти. Змінивши стан форми, наприклад вибравши якесь значення в елементі select та натиснувши на кнопку спрацьовує event handler (обробник подій), який викликає необхідний метод, який в свою чергу відправляє SPARQL запит до відповідного онтологічного файлу. Знайдені дані виводяться користувачу в поле dataGridView. У випадку, якщо даних не було знайдено, виводиться відповідне модальне вікно з повідомленням.

З отриманими даними користувач також може взаємодіяти, наприклад подвійний клік на рядок з результатом дає можливість переглянути детальнішу інформацію (якщо це навчальна дисципліна, або відкрити файл (якщо це методичний матеріал).

ВИКОРИСТУВУВАНІ ТЕХНІЧНІ ЗАСОБИ

Для створення даної програми було використано середовище розробки Visual Studio 2017 та мова програмування C# з використанням .Net Framework та бібліотеки RDF.

Для створення та редагування онтологій було розглянуто декілька редакторів, про те вибрано було програму Protégé 5.5.0.

Для запуску даної програми користувачу необхідно мати встановлений на своєму комп’ютері .Net Framework.

ВХІДНІ ТА ВИХІДНІ ДАНІ

Вхідними даними є:

- онтологічний файл формату .owl під назвою curriculum.owl;
- онтологічний файл формату .owl під назвою documents.owl;
- дані про методичні матеріали та навчальні дисципліни.

Вихідними даними є:

- виведення інформації виконаного SPARQL запиту до онтології;
- інформація про навчальні дисципліни;
- ссилка на файл з методичним матеріалом.

ДОДАТОК 4

Онтологічна система інформаційних ресурсів кафедри

Апробація

УКР.НТУУ«КПІ ім. Ігоря Сікорського»_ТЕФ_АПЕПС_ТР61110_20Б

Аркушів 4

Київ – 2020

Було подано тези на факультетську конференцію «Сучасні проблеми наукового забезпечення енергетики». Тези надруковано в другому томі, в дев'ятій секції на сторінці 101. Текст тез наведено нижче.

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

СУЧАСНІ ПРОБЛЕМИ НАУКОВОГО ЗАБЕЗПЕЧЕННЯ ЕНЕРГЕТИКИ

Матеріали XVIII Міжнародної науково-практичної конференції молодих вчених і студентів 2020 року

ТОМ 2

НТУУ "КПІ" 1931 ТЕФ

Київ- 2020

УДК 004.045

Студент 4 курсу, гр. ТР-61 Круглий Д.В.
Ст.викл. Дацюк О.А.

ОНТОЛОГІЧНА ПОБУДОВА РЕЄСТРУ НАВЧАЛЬНИХ ПЛАНІВ ФАКУЛЬТЕТУ

Аналізуючи навчальні плани, ми аналізуємо достатньо об'ємну предметну область, до складу якої входить велика кількість даних різних форматів та типів, завдяки чому аналіз стає досить складною та об'ємною процедурою, тому для спрощення обробки семантичних зв'язків між сутностями предметної області досить часто використовуються онтології. Щодо семантичних зв'язків, то ними називають значущі асоціації між двома або більше поняттями або наборами сутностей [1].

Одним із способів розв'язання проблеми розподіленої інформації є побудова окремої інформаційної системи для кожної гілки організації. Для створення реєстру навчальних планів факультету необхідний неперервний системний аналіз інформаційних ресурсів, а особливо їх взаємозв'язків і захищеності. До того ж, варто зауважити, що реєстр слід розглядати як розподілену складну інформаційну систему, тому що він наділений всіма властивостями складних систем, тобто: великою кількістю складових компонентів, взаємодією з навколишнім середовищем. Неможливо також не згадати про ієрархічну структуру та мінливість у часі. Відомий W3 стандарт OWL – мова Web онтологій надає засоби для подання та пов'язування онтологій у Web просторі в форматі, що є «зрозумілим» машиною [2].

Система реалізована мовою програмування JavaScript, може бути доступною за посиланням кожному користувачеві. До основних переваг даного продукту можна віднести:

- актуальність програми;
- простий і зручний інтерфейс для користувача;
- формальна структура онтологічного файлу, яка значно спрощує обробку;
- можливість автоматичного створення зв'язків між таблицями;
- швидке представлення даних користувачеві за запитом.

Результатом упровадження програми стане можливість акумуляції освітніх ресурсів факультету в одній розподіленій системі. Такий підхід надасть можливість у режимі реального часу отримувати актуальну, несуперечливу інформацію про освітні ресурси, а також автоматизувати формування звітної документації. Розробка розподіленої архітектури дозволить рівномірно розподілити навантаження на онтологічну базу. Отже створюється унікальний програмний продукт, який дозволяє, проаналізувавши навчальний план, знайти необхідну користувачеві інформацію. Застосування програми досить різноманітне: це і пошук необхідної інформації для студентів, а також полегшення роботи для викладачів.

Перелік посилань:

1. Zhang J. Ontology and the Semantic Web / Jane Zhang. // Proceedings of the North American Symposium on Knowledge Organization. – 2007. – №1.
2. OWL Specification Development . [Електронний ресурс] // Режим доступу: https://www.w3.org/2001/sw/wiki/OWL