

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістр

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ
АВТОМАТИЧНОЇ КЛАСИФІКАЦІЇ МЕДИЧНИХ ПРОТИПОКАЗАНЬ
RESEARCH AND SOFTWARE IMPLEMENTATION OF METHODS FOR
AUTOMATIC CLASSIFICATION OF MEDICAL CONTRAINDICATIONS

Виконав(ла): студент(ка) 2 курсу, групи КНТ-142м

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

КОСТЕНКО А.А.

(ПРИЗВИЩЕ та ініціали)

Керівник КОЦУР М.І.

(ПРИЗВИЩЕ та ініціали)

Рецензент СКРУПСЬКИЙ С.Ю.

(ПРИЗВИЩЕ та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
(повне найменування закладу вищої освіти)

Факультет КНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 121 Інженерія програмного забезпечення
(код і найменування)
Освітня програма (спеціалізація) Інженерія програмного забезпечення
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
“ ” 2023 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

КОСТЕНКА Андрія Андрійовича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів автоматичної класифікації медичних протипоказань. Research and Software Implementation of Methods for Automatic Classification of Medical Contraindications

керівник проєкту (роботи) к.т.н., доцент, КОЦУР Михайло Ігорович,
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “14” листопада 2023 року № 437

2. Строк подання студентом проєкту (роботи) 01 грудня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Дослідження методів автоматичної класифікації медичних протипоказань. 3. Програмна реалізація методу. 4. Тестування методів автоматичної класифікації медичних протипоказань.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	КОЦУР М.І., доцент		
Нормоконтроль	ЛИПОВЕЦЬ М.В., асистент		

7. Дата видачі завдання “08” вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітки
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Розробка та удосконалення методів, моделей й алгоритмів вирішення задачі.	2-3 тижні	Розділ 2
4	Розробка онтології предметної області.	4-5 тижні	Розділ 3
5	Розробка додатку.	6-7 тижні	
6	Тестування методів класифікації.	8 тиждень	Розділ 4
7	Оформлення пояснювальної записки та документів до неї.	9-10 тижні	Додатки
8	Нормоконтроль та рецензування.	11 тиждень	
9	Захист роботи.	12 тиждень	

Студент(ка)

(підпис) Андрій КОСТЕНКО
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Михайло КОЦУР
(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
90 с., 9 табл., 19 рис., 3 дод., 42 джерела.

МЕДИЧНІ ПРОТИПОКАЗАННЯ, АВТОМАТИЧНА КЛАСИФІКАЦІЯ,
МАШИННЕ НАВЧАННЯ, ОНТОЛОГІЇ, ПРОГРАМНА РЕАЛІЗАЦІЯ,
ТЕСТУВАННЯ.

Об'єкт дослідження – процес автоматичної класифікації медичних протипоказань.

Предмет дослідження – методи та засоби автоматичної класифікації медичних протипоказань.

Мета роботи – підвищення ефективності автоматичної класифікації медичних протипоказань на основі удосконалення методів та розробки програмного забезпечення.

Матеріали, методи та технічні засоби: методи класифікації даних, онтологічне моделювання предметної області, об'єктно-орієнтоване програмування, мови програмування Python, OWL, SPARQL, SQL, персональний комп'ютер з процесором Intel Core 2 Duo під управлінням операційної системи Microsoft Windows 11.

Наукова новизна. Удосконалено метод автоматичної класифікації медичних протипоказань на основі інтеграції онтологій та машинного навчання.

Результати. Розроблено програмне забезпечення для автоматичної класифікації медичних протипоказань на основі онтологій.

Галузь використання – медичні заклади.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 90 pages, 9 tables, 19 figures, 3 appendices, 42 sources.

MEDICAL CONTRAINDICATIONS, AUTOMATIC CLASSIFICATION, MACHINE LEARNING, ONTOLOGIES, SOFTWARE IMPLEMENTATION, TESTING.

The object of the research is the process of automatic classification of medical contraindications.

The subject of research is methods and means of automatic classification of medical contraindications.

The purpose of the work is to increase the efficiency of the automatic classification of medical contraindications based on the improvement of methods and software development.

Materials, methods and technical means: methods of data classification, ontological modeling of the subject area, object-oriented programming, programming languages Python, OWL, SPARQL, SQL, a personal computer with an Intel Core 2 Duo processor running the Microsoft Windows 11 operating system .

Scientific novelty. The method of automatic classification of medical contraindications based on the integration of ontologies and machine learning has been improved.

The results. Software for automatic classification of medical contraindications based on ontologies has been developed.

Field of use - medical facilities.

ЗМІСТ

	С.
Перелік скорочень і умовних позначок	7
Вступ	8
1 Аналіз предметної області	10
1.1 Поняття медичних протипоказань	10
1.2 Методи класифікації даних	12
1.3 Постановка завдань розробки програмного забезпечення	15
1.4 Висновки до розділу 1	17
2 Дослідження методів автоматичної класифікації медичних протипоказань	18
2.1 Предметна область онтологій	18
2.2 Особливості онтологій в сфері медицини	21
2.3 Інтерфейси програмування для онтологій	26
2.4 Порівняння об'єктних моделей і формальних онтологій	31
2.5 Висновки до розділу 2	33
3 Програмна реалізація методу	34
3.1 Структурна схема програмної реалізації методів	34
3.2 Модель OWL на Python	37
3.3 Функції доступу до онтології	42
3.4 Висновки до розділу 3	44
4 Тестування методів автоматичної класифікації медичних протипоказань	45
4.1 Автоматична класифікація та міркування	45
4.2 Експериментальне дослідження	50
4.3 Висновки за розділом	52
Висновки	53
Перелік джерел посилань	54
Додаток А Технічне завдання	69
Додаток Б Онтологія медичної предметної області	73
Додаток В Слайди презентації	85

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

AI	– Artificial Intelligence;
ANN	– Artificial Neural Network;
API	– Application Programming Interface;
CPU	– Central Processing Unit;
DL	– Deep Learning;
IDE	– Integrated Development Environment;
GPU	– Graphics Processing Unit;
ML	– Machine Learning;
NLP	– Natural Language Processing;
OWL	– Web Ontology Language;
RDF	– Resource Description Framework;
SPARQL	– SPARQL Protocol and RDF Query Language;
SVM	– Support Vector Machine.

ВСТУП

Сучасний стан розробки програмного забезпечення для дослідження та впровадження методів автоматичної класифікації медичних протипоказань глибоко переплітається з досягненнями в галузі штучного інтелекту (AI), машинного навчання (ML) та онтологій. Ці інновації зробили великий прогрес в програмному забезпеченні охорони здоров'я, забезпечивши більш ефективний та персоналізований догляд за пацієнтами. Ці технології розширюють можливості прогностичної аналітики, дозволяючи проактивно керувати здоров'ям і складати більш індивідуальні плани лікування [1].

Дані про стан здоров'я, створені пацієнтами, набувають все більшого значення, що відображає тенденцію до догляду, орієнтованого на пацієнта. Однак такі проблеми, як безпека даних, конфіденційність, відповідність нормативним вимогам і сумісність, залишаються серйозними перешкодами, які необхідно вирішити, щоб максимізувати переваги цих технологій.

Дана галузь швидко розвивається завдяки технологічним інноваціям, які змінюють способи класифікації та управління медичними протипоказаннями, хоча й не обходиться без серйозних недоліків, які потребують пильної уваги та вирішення [2].

Програмне забезпечення для реалізації методів автоматичної класифікації медичних протипоказань першу чергу ґрунтується на інтелектуальному аналізі даних. Сервіс, орієнтований на онтологію інструмент програмування на Python, особливо корисний для біомедичних онтологій. Цей інструмент є значним прогресом в інтеграції онтологічних моделей із більш традиційним підходом до об'єктно-орієнтованого програмування, пропонуючи унікальні та потужні засоби для керування складними біомедичними даними.

Програмне забезпечення на основі онтологій дозволяє програмістам Python маніпулювати предметними сутностями, використовуючи знайомий синтаксис і парадигми об'єктно-орієнтованого програмування, тим самим покращуючи легкість доступу та модифікації цих сутностей. Ця повна

інтеграція є особливо вигідною для користувачів, які знайомі з Python, але меншою мірою для користувачів, знайомих із тонкощами онтологічних моделей або мов, таких як OWL (мова вебонтології) [3].

Однак цей підхід має недоліки. Складність представлення медичних концепцій і зв'язків в онтологіях, які часто вимагають рівня точності та деталізації, якими може бути важко керувати, становить значну перешкоду. Крім того, припущення про відкритий світ, який лежить в основі багатьох онтологічних моделей, може суперечити підходу, заснованому на доказах, який є центральним у сучасній медицині, створюючи потенційні проблеми в тому, як медичні знання представлені та використовуються.

У ширшому контексті онтологій, ця сфера зазнала значного зростання, особливо в біомедичних додатках. Онтології все частіше отримують визнання за їхню здатність структурувати великі та складні набори даних, сприяючи взаємодії та розширюючи можливості аналізу даних і аргументації. Розробляємий сервіс спрямований на подолання розриву між традиційними мовами програмування та онтологічними моделями, роблячи ці потужні інструменти більш доступними для ширшого кола користувачів.

Розробка та вдосконалення таких інструментів дуже актуальні в розвитку сучасних сервісів, що керується даними, особливо в біомедичній галузі. Ці інструменти не тільки сприяють більш ефективному управлінню та аналізу даних, але й мають потенціал для значного просування досліджень і надання медичної допомоги. Спрощуючи взаємодію зі складними онтологічними моделями, вони дають змогу ширшому колу професіоналів взаємодіяти з цими моделями та використовувати їхні можливості, таким чином стимулюючи інновації та покращуючи результати в різних сферах.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття медичних протипоказань

Медичні протипоказання стосуються конкретних ситуацій, у яких не слід застосовувати препарат, процедуру чи операцію, оскільки це може бути шкідливим для пацієнта. Ці протипоказання можуть бути абсолютними, коли ризик побічних ефектів високий і однозначно переважає будь-яку терапевтичну користь, або відносними, коли ризики можуть бути переважені користю за певних обставин [4].

В таблиці 1.1 описані параметри та вимоги використання програмних засобів для зберігання даних медичних протипоказань.

Таблиця 1.1 – Зберігання даних на програмному рівні

Параметр	Опис
Дизайн бази даних	Програмне забезпечення має використовувати надійну базу даних для зберігання даних протипоказань. Ця база даних включатиме таблиці для ліків, медичних умов, демографічних даних пацієнтів і особливостей протипоказань.
Нормалізація даних	Забезпечення узгодженості даних шляхом нормалізації бази даних для зменшення надмірності та залежності.
Безпека та відповідність	Впровадження надійних протоколів безпеки та дотримання правил охорони здоров'я, таких як HIPAA, для захисту конфіденційної інформації пацієнтів.

Медичні протипоказання – це конкретні обставини, за яких лікарський засіб, процедура, хірургічне втручання чи інше лікування вважаються недоцільними, оскільки вони можуть бути шкідливими для пацієнта. Вони виникають внаслідок різних факторів, таких як історія хвороби пацієнта, алергія, поточні ліки, вік, стан вагітності або певні стани здоров'я. Концепція ґрунтується на принципі «по-перше, не нашкодь», гарантуючи, що проведене

лікування не наражає пацієнта на невиправданий ризик.

У контексті інтелектуальної системи для керування медичними протипоказаннями ці деталі будуть ретельно записуватися та інтегруватися в цифровий профіль здоров'я пацієнта. Ця система використовує бази даних і алгоритми для перехресних посилань на інформацію про здоров'я пацієнта з величезним сховищем медичних знань про ліки, процедури та умови. Коли постачальник медичних послуг розглядає можливість лікування, система автоматично аналізує інформацію пацієнта за цією базою даних.

Ядром такої системи є динамічна та постійно оновлювана база даних, яка зберігає детальну інформацію про різні види лікування та їх можливі протипоказання. Ця база даних є не просто статичним сховищем інформації; він розроблений, щоб бути інтерактивним і чуйним. Коли медичний працівник вводить запропоноване лікування, система порівнює збережені медичні дані пацієнта зі своєю базою даних. Якщо виявлено протипоказання, система попереджає провайдера, пропонуючи детальну інформацію про характер ризику та, в ідеалі, пропонуючи альтернативні методи лікування, які є безпечнішими для конкретного пацієнта.

Щоб забезпечити точність і релевантність, система має бути заповнена останніми медичними дослідженнями та рекомендаціями. Інтеграція з електронними медичними записами (EHR) є надзвичайно важливою, оскільки вона дозволяє оновлювати профілі пацієнтів у реальному часі. Безпека та конфіденційність є найважливішими, враховуючи конфіденційність медичних даних. Система використовуватиме надійне шифрування та механізми контролю доступу для захисту інформації пацієнтів.

Підводячи підсумок, можна сказати, що інтелектуальна система для медичних протипоказань є значним прогресом у персоналізованій медицині, яка спрямована на підвищення безпеки пацієнтів, надаючи медичним працівникам миттєве уявлення на основі даних про потенційні ризики різних медичних процедур для окремих пацієнтів.

Створення сервісу для класифікації медичних протипоказань передбачає

глибоке розуміння медичних протипоказань, надійні та безпечні рішення для зберігання даних та ефективне використання цих даних через зручний інтерфейс, інтеграцію з наявними медичними записами та застосування передових технологій обробки даних (табл. 1.2).

Таблиця 1.2 - Використання даних про медичні протипоказання

Параметр	Опис
Інтерфейс користувача (UI)	Розробити зручний інтерфейс користувача, де медичні працівники можуть вводити інформацію про пацієнта та отримувати сповіщення про протипоказання
Інтеграція з EHR	Інтеграція з існуючими електронними медичними записами (EHR) для безперебійного потоку даних і доступу
Алгоритми обробки даних	Для аналізу даних пацієнтів із базою даних протипоказань, надаючи персоналізовані попередження щодо протипоказань у реальному часі або прогнозуючому часі
Механізм оновлення	Механізм для регулярного оновлення бази даних новими медичними висновками та інформацією про ліки
Звітування та аналітика	Інструменти звітності для перегляду випадків і моделей протипоказань, допомагаючи приймати кращі рішення

1.2 Методи класифікації даних

Класифікація медичних даних – це складний процес, який передбачає організацію різних типів даних, починаючи від історій пацієнтів і результатів лабораторних досліджень до досліджень зображень і генетичної інформації.

В основі цього процесу лежить створення стандартизованих систем кодування, таких як Міжнародна класифікація хвороб (МКБ) для діагностики та Поточна процедурна термінологія (CPT) для медичних процедур. Ці системи кодування забезпечують універсальну мову для документування та розуміння медичних станів і методів лікування, забезпечуючи узгоджену комунікацію між

різними системами охорони здоров'я та фахівцями.

На додаток до цих систем кодування, класифікація медичних даних часто включає класифікацію даних на основі їх природи та джерела. Наприклад, демографічна інформація про пацієнта, історія хвороби, симптоми, діагнози, плани лікування та результати – все чітко класифіковано. Ця категоризація полегшує здатність інтелектуальної системи швидко отримувати доступ і аналізувати певні типи даних, що стосуються конкретної ситуації, наприклад, ідентифікувати потенційні протипоказання.

Крім того, класифікація поширюється на стратифікацію даних на основі тяжкості, терміновості та інших клінічно значущих параметрів. Наприклад, гострі стани можуть позначатися інакше, ніж хронічні, а надзвичайним ситуаціям надається пріоритет під час пошуку й аналізу даних.

У контексті протипоказань система класифікації буде розроблена для визначення та виділення критичних даних, які можуть вплинути на рішення щодо лікування. Це включає алергію, раніше задокументовані побічні реакції на ліки, супутні захворювання та поточні ліки, які можуть негативно взаємодіяти з запропонованими методами лікування.

Інтеграція машинного навчання та штучного інтелекту в цю структуру покращує здатність системи розпізнавати закономірності та кореляції в медичних даних, які можуть бути не відразу очевидними для спостерігачів. Ці технології можуть допомогти передбачити ризики та результати на основі складних наборів даних, пропонуючи неоціненну підтримку в процесах прийняття рішень.

Інструменти візуалізації даних також відіграють вирішальну роль, перетворюючи складні медичні дані на зрозумілу та практичну інформацію. Ці інструменти дозволяють постачальникам медичних послуг швидко зрозуміти стан здоров'я та історію пацієнта, допомагаючи визначити потенційні протипоказання.

Сучасні методи виходять за рамки традиційних баз даних і систем зберігання, зокрема, завдяки інтеграції онтологій для зберігання медичних

даних.

Онтології в контексті медичних даних забезпечують структуровану та семантичну структуру, яка представляє складні відносини між різними медичними концепціями. Цей підхід є особливо вигідним для класифікації медичних протипоказань, оскільки він дозволяє представити складні взаємозв'язки між ліками, захворюваннями, демографічними характеристиками пацієнтів та іншими важливими факторами. Онтології дозволяють системі розуміти та інтерпретувати медичний контекст за межі простих точок даних, пропонуючи більш тонке розуміння протипоказань [5].

Хмарні системи зберігання даних стають все більш популярними, пропонуючи масштабованість, гнучкість і доступність. Ці системи можуть зберігати величезні обсяги даних, отриманих із різних джерел, включаючи електронні медичні записи (EHR), портали пацієнтів та пристрої IoT. Хмарне сховище також забезпечує обробку даних у режимі реального часу та доступ до них, що має вирішальне значення для служби, яка покладається на актуальну інформацію для точної класифікації.

Щоб класифікувати дані, використовуються алгоритми машинного навчання і штучного інтелекту. Ці алгоритми можуть аналізувати великі набори даних, щоб визначити потенційні протипоказання, вивчаючи історичні дані. Вони можуть виявити закономірності та кореляції, які можуть бути очевидними не відразу, підвищуючи точність прогнозів щодо протипоказань [6].

Обробка природної мови (NLP), підмножина штучного інтелекту, відіграє вирішальну роль в інтерпретації неструктурованих даних, таких як клінічні нотатки та дослідницькі статті. Алгоритми NLP можуть витягувати з цих текстів релевантну інформацію, необхідну для повного розуміння потенційних протипоказань [7].

Інтеграція з існуючими системами EHR також є ключовим аспектом. Ця інтеграція забезпечує доступ до останніх даних про пацієнтів, що необхідно для точної класифікації протипоказань. Це також забезпечує безперебійний потік даних між різними системами охорони здоров'я, підвищуючи ефективність

медичних робочих процесів.

Інструменти візуалізації даних можуть покращити розуміння складних наборів даних. Ці інструменти можуть представляти результати класифікації в зручному для сприйняття форматі, допомагаючи медичним працівникам швидко визначати ключові ідеї [1].

Важливий механізм постійного навчання. Система має бути розроблена таким чином, щоб з часом розвиватися та вдосконалюватися, враховуючи відгуки та результати її класифікацій. Такий підхід гарантує, що служба залишається в курсі останніх медичних знахідок і адаптується до мінливого середовища охорони здоров'я.

Розробка сервісу для класифікації медичних протипоказань передбачає використання онтологій для розуміння нюансів даних, використання хмарного сховища для гнучкості, застосування AI та ML для класифікації даних, інтеграцію NLP для обробки неструктурованих даних, забезпечення сумісності EHR, розробку зручних інтерфейсів, використання інструментів візуалізації даних і включення механізму безперервного навчання. Ці елементи разом створюють складну та ефективну систему, здатну впоратися зі складністю класифікації медичних протипоказань.

1.3 Постановка завдань розробки програмного забезпечення

Створення служби класифікації медичних протипоказань як розробник програмного забезпечення передбачає структурований процес, який починається з концептуалізації та планування. На цьому початковому етапі визначаються обсяг і цілі проєкту, зосереджуючись на розумінні конкретних потреб для класифікації медичних протипоказань. Після цього відбувається збір вимог, де детальні вимоги збираються з доступних для широкого загалу даних [8].

Наступним етапом є проєктування та архітектура, яка передбачає проєктування архітектури програмного забезпечення, включаючи інтерфейс

користувача, схему бази даних і спосіб інтеграції системи з іншими системами. Після того, як дизайн завершено, починається етап розробки, що включає кодування зовнішніх і внутрішніх компонентів програмного забезпечення [4].

Після розробки необхідні ретельне тестування та гарантія якості. Цей етап передбачає перевірку програмного забезпечення на наявність помилок, забезпечення точності класифікації протипоказань і оцінку зручності використання, щоб гарантувати, що програмне забезпечення відповідає потребам користувачів і галузевим стандартам. В таблиці 1.3 описані запропоновані задачі дослідження та розробки.

Таблиця 1.3 – Етапи дослідження

Задача	Опис
Аналіз вимог	Розуміти типи протипоказань, які необхідно класифікувати
Збір даних і керування ними:	Використовувати відкриті джерела
Вибір технологій	Провести аналіз технологій автоматичної обробки даних
Розробка онтології	Використання онтологій для структурування медичних даних, що дозволяє краще класифікувати та розуміти складні медичні зв'язки
Розробка програмного забезпечення	Розробка програми з акцентом на зручність для користувача інтерфейси
Тестування та перевірка	Щоб забезпечити точність і надійність.

Завдяки ретельному розгляду кожного кроку та інтеграції відповідних технологій буде розроблено ефективний сервіс для класифікації медичних протипоказань, що покращить прийняття медичних рішень і допомогу пацієнтам.

1.4 Висновки до розділу 1

У першому розділі проаналізовано предметну область дослідження - медичні протипоказання та методи їх автоматичної класифікації. Визначено об'єкт, предмет та мету дослідження. Сформульовано вимоги до програмного забезпечення та основні задачі його розробки.

Отримані результати дозволяють сфокусувати подальші дослідження на розробці та вдосконаленні методів автоматичної класифікації медичних протипоказань з використанням сучасних підходів та технологій штучного інтелекту. Запропоновано шляхи підвищення ефективності такої класифікації та її програмної реалізації.

2 ДОСЛІДЖЕННЯ МЕТОДІВ АВТОМАТИЧНОЇ КЛАСИФІКАЦІЇ МЕДИЧНИХ ПРОТИПОКАЗАНЬ

2.1 Предметна область онтологій

Формальна онтологія – це специфікація понять, їхніх атрибутів і зв’язків у певній області дискурсу [1]. Онтології мають два головних інтереси:

- вони можуть бути використані для виконання логічних висновків для виведення нових фактів, з логічним обґрунтуванням;
- вони можуть зв’язувати різні частини знань з різних онтологій у семантичній мережі.

W3C (World Wide Web Consortium) запропонував OWL (Web Ontology Language) [2] для формалізації онтологій, а онтології OWL часто представлені у вигляді графів RDF (Resource Description Framework) [3].

Онтології, в контексті інформатики та штучного інтелекту, є формальними представленнями знань із багатим набором концепцій і зв’язків, характерних для конкретної області. Вони забезпечують структуру для розуміння та значущої обробки даних не лише людьми, а й машинами.

За останні роки у розробці та застосуванні онтологій відбувся значний прогрес. Однією з ключових тенденцій є зростаюча витонченість того, як конструюються та використовуються онтології. Зі зростанням обсягу великих даних і більш складних інформаційних потреб онтології еволюціонували від простих ієрархічних структур до більш складних мереж, здатних представляти складні взаємозв’язки та залежності всередині даних.

Інтеграція онтологій з машинним навчанням і технологіями обробки природної мови була головною сферою прогресу. Ця інтеграція дозволяє системам не тільки розуміти структуровані дані, але й інтерпретувати й обробляти неструктуровані дані, такі як текст і мова. У результаті інтелектуальні системи стають більш вправними у розумінні контексту, семантики та нюансів у людській мові, що значно покращує їхню застосовність у таких сферах, як семантичний пошук, пошук інформації та системи підтримки

прийняття рішень.

Іншою важливою подією є рух до більш динамічних та адаптивних онтологій. Традиційно онтології були відносно статичними, вимагаючи ручного оновлення та перегляду. Проте з розвитком штучного інтелекту зростає тенденція до онтологій, які можуть розвиватися та адаптуватися з часом, навчаючись на нових даних і взаємодії з користувачем. Ця адаптивність має вирішальне значення для збереження актуальності та точності онтології в таких галузях, які швидко змінюються, як медицина, фінанси та технології.

Підвищена увага приділяється сумісності та стандартизації. Оскільки використання онтологій розширюється в різних областях і галузях, потреба в стандартних форматах і протоколах стає більш критичною. Це гарантує, що онтології, розроблені в різних контекстах, можуть спілкуватися та працювати разом, забезпечуючи більш плавну інтеграцію та обмін даними між системами.

Співпраця та розробка з відкритим кодом також стали більш поширеними. Зростає визнання того, що розробка комплексних і точних онтологій може бути масштабною справою, яка часто виходить за рамки однієї організації. Тому спільні зусилля, часто у формі проєктів з відкритим кодом, стають все більш поширеними, об'єднуючи ресурси та досвід з усього світу.

Сфера онтологій швидко розвивається, відзначаючись прогресом у складності, інтеграції з передовими технологіями, адаптивністю, стандартизацією та спільною розробкою. Ці тенденції не тільки розширюють можливості інтелектуальних систем, але й відкривають нові можливості для того, як ми керуємо та використовуємо знання в різних областях.

Онтології використовувалися в багатьох областях, зокрема в біомедичній сфері [4], [5]. Сьогодні це одна з найскладніших областей людських знань, і існує зростаюча потреба в структуруванні та формалізації цих знань. Понад двадцять років тому проєкту GALEN [6] вже базувався на онтології. На розвиток останніх біомедичних термінологій також сильно вплинули онтології. Прикладами є SNOMED CT (Стандартизована номенклатура медицини – клінічні терміни) [7], генна онтологія [8] і FMA

(Основна модель анатомії) [9], який нещодавно було перетворено на OWL 2 [10].

Було запропоновано багато методів та інструментів для проєктування, видання, підтримки, вирівнювання або оцінки онтологій, включаючи редактор Protégé та аргумент HermiT [11]. Програмні інтерфейси онтологій – це інструменти для доступу та модифікації онтології в межах мови програмування. Типові приклади використання включають заповнення онтології даними, витягнутими з бази даних, або створення вебсайту, на якому відображаються висновки, зроблені за допомогою резонера в онтології. Мови програмування, особливо об'єктно-орієнтовані мови програмування, широко викладаються та використовуються в наш час. Таким чином, це загальна потреба в інтерфейсі онтологій з цими мовами програмування.

У літературі запропоновано декілька стратегій для інтерфейсів програмування онтології мови запитів, такі як SPARQL, API (інтерфейси прикладного програмування), такі як OWLAPI, і онтологічно-орієнтоване програмування. Зазвичай використовуються перші дві стратегії. Однак із цими стратегіями працювати з онтологіями набагато довше й складніше, ніж розробляти об'єктно-орієнтовану програму. Остання стратегія все ще є предметом дослідження, але вона також виглядає найбільш перспективною [12].

Прикладом може служити порівняння протипоказань між кількома препаратами. Протипоказання можна отримати з баз даних лікарських засобів або ввести експертом вручну з офіційних довідкових документів. Однак через синонімічність, розділення та варіації рівня деталізації протипоказання не можна порівнювати «як є» між препаратами. Наприклад, препарат А може бути протипоказаний при «набутому геморагічному розладі» та «фульмінантному гепатиті», а препарат В – при «геморагічному розладі» та «тяжкій печінковій недостатності», але це не означає, що препарат В не протипоказаний у випадку «набутого геморагічного розладу» та «фульмінантного гепатиту». Рішення полягає в тому, щоб розробити онтологію та використовувати резонер [13].

Протипоказання та розлади не можуть бути представлені окремими особами в онтології. Вони можуть бути представлені лише класами через зв'язки підпорядкування, які існують між ними. Але класами важче маніпулювати за допомогою звичайних API. По-друге, обґрунтування має враховувати лише відомі протипоказання (тобто наявні в базах даних або внесені експертом). Це вимагає локально закритого світового міркування, яке не підтримується звичайними міркуваннями OWL [14].

Цілі цієї роботи полягали в розробці модуля для онтологічно-орієнтованого програмування на Python з підтримкою онтологій OWL для вирішення двох задач:

- забезпечення чіткого, лаконічного та легкого у використанні синтаксису, заснованого на нотаціях редактора онтології Protégé, так і на нотації з крапками, загальною для багатьох об'єктно-орієнтованих мов програмування, включаючи Python;
- надання синтаксичних елементів високого рівня для полегшення маніпулювання класами та простого алгоритму для локальних міркувань у закритому світі.

2.2 Особливості онтологій в сфері медицини

З нашого досвіду в цій галузі ми визначили дві особливості медичних знань, які ускладнюють розробку онтологій і онтологічних міркувань. Вони описані в двох наступних підрозділах.

У медицині знання, наскільки це можливо, базуються на доказах, тобто знання походять із доказу, як правило, значної різниці, яка спостерігається під час клінічного випробування та включає статистичне тестування. Це відомо як доказова медицина (EBM) [15]. Класичний дизайн клінічного випробування полягає у наборі двох групах пацієнтів, кожна група отримує різні лікування (наприклад, препарат А або ефективність лікування для кожного пацієнта з урахуванням біологічних маркерів або еволюції симптомів. Якщо різниця між

ефективністю двох груп є статистично значущою, один із двох препаратів є більш ефективним і йому слід віддати перевагу; це породжує нову частину медичних знань. За відсутності доказів (і тільки в цьому випадку) медичні знання повинні бути встановлені шляхом врахування консенсусної думки груп експертів.

Отже, у медичному обґрунтуванні правдивими слід вважати лише знання, що ґрунтуються на доказах або підтверджені експертами. Таким чином, припущення відкритого світу, яке вважає, що все, що не сказано, можливе і може бути правдою, не підходить для медичних знань. Міркування не повинні будувати нові гіпотези щодо медичних знань, оскільки така гіпотеза не буде ґрунтуватися на доказах. Навпаки, припущення відкритого світу є цікавим, якщо розглядати дані пацієнтів. Це дозволяє досліднику будувати гіпотези щодо ще невідомих розладів або клінічних станів пацієнта. Це особливо корисно в діагностичних системах [16].

Наприклад, коли лікар перевіряє протипоказання до препарату, він вважає, що всі протипоказання відомі (і перераховані в офіційних текстах і базах даних ліків), навіть якщо це може бути не зовсім так. Таким чином, система підтримки прийняття медичних рішень повинна міркувати лише з використанням відомих протипоказань, а не будувати гіпотези щодо потенційних нових протипоказань. Це призведе до дурних попереджень на кшталт «пацієнт діабетик, препарат, який ви призначаєте, не протипоказаний при діабеті, але може існувати ще невідоме протипоказання». Перевірте свій рецепт!» [17].

Підсумовуючи, для медичного обґрунтування необхідне поєднання припущень відкритого та закритого світу. Це зазвичай називають міркуванням локального закритого світу. Інші автори [16] зробили подібне спостереження, міркуючи про записи пацієнтів та залучення пацієнтів до клінічних випробувань, і вони показали, що деякі дані пацієнтів слід розглядати за припущенням відкритого світу, а інші – ні. Діагностику пацієнтів вони вважали «відкритою» (можуть бути ще невідомі діагностики), а аптечні дані пацієнтів –

«закритими» (усі ліки, призначені пацієнту, відомі через аптечну службу).

Зазвичай розглядається багато рівнів деталізації. Наприклад, розлади можуть бути виражені з різним ступенем деталізації, наприклад, запальні розлади кишечника, розлад Крона, важкий розлад Крона, розлад Крона з шкірними проявами тощо [17]. Дані пацієнта, як правило, описуються з дрібною деталізацією, наприклад «М. Х страждає на розлад Крона», тоді як медичні знання описані на більш загальному рівні, наприклад, «цей препарат протипоказаний із запальними захворюваннями кишечника» (тобто включаючи розлад Крона). Медикаментозне лікування поводитьсь подібним чином, наприклад, анальгетики, нестероїдні протизапальні препарати (НПЗП), аспірин, аспірин у антитромбоцитарних показаннях, аспірин у таблетках 1 г тощо. Ці численні рівні деталізації ускладнюють представлення медичних концепцій, тому що при розгляді медичне поняття, майже завжди можна конкретизувати його для визначення більш тонкого поняття.

В онтологіях рівні деталізації представлені за допомогою відношень is-a. Оскільки між індивідами не існує зв'язків між собою, основні медичні поняття, такі як розлади та медикаментозне лікування, можуть бути представлені лише класами, а не індивідами [18].

Онтології в медицині служать структурованою основою для представлення складних медичних знань, включаючи хвороби, симптоми, лікування та навіть складні взаємозв'язки між цими елементами. Ці онтології мають ключове значення для забезпечення складного аналізу даних, підтримки прийняття рішень і досліджень у сферах медицини [19].

Основною характеристикою медичних онтологій є їх детальний і вичерпний характер. Їм потрібно точно охопити широкий спектр медичних концепцій та їх взаємозв'язки. Це включає не лише хвороби та симптоми, а й анатомічну інформацію, біохімічні елементи та процедурні деталі. Здатність представити цю різноманітну та складну інформацію в структурованому вигляді забезпечує кращу взаємодію даних, необхідну для різних медичних програм, таких як електронні записи про стан здоров'я (EHR), системи

підтримки клінічних рішень і бази даних медичних досліджень.

Ще одна важлива особливість – це потреба в точності та стандартизації. Медичні онтології повинні відповідати високим стандартам точності та послідовності, оскільки вони використовуються в контекстах, де помилки чи неясності можуть мати серйозні наслідки. Ця точність поширюється на необхідність відповідності міжнародним медичним стандартам і термінології, таким як Міжнародна класифікація хвороб (МКБ) і Систематизована номенклатура лікарських засобів – клінічні терміни (SNOMED CT) [31].

Сумісність також є ключовою характеристикою. Враховуючи різноманітність систем і методів охорони здоров'я, медичні онтології необхідно розробляти таким чином, щоб їх можна було легко інтегрувати та використовувати в різних системах і платформах. Ця сумісність сприяє безперебійному обміну даними та співпраці між різними установами та системами охорони здоров'я.

Ключові приклади розроблених онтологій включають SNOMED CT, який надає вичерпну клінічну термінологію, фундаментальну модель анатомії (FMA), що зосереджується на анатомії людини, і генну онтологію (GO), яка має вирішальне значення для генетичних досліджень. Кожна з цих онтологій служить різним, але збігаючимся цілям у сфері медицини, від клінічної допомоги до досліджень і геноміки [32].

Медичні онтології є наріжним каменем у сфері інтелектуальних медичних систем, забезпечуючи необхідну структуру та стандартизацію для обробки складних медичних даних. Вони характеризуються своєю глибиною, точністю, сумісністю та адаптивністю, кожен з яких адаптований до конкретних аспектів медичних знань і практики. У міру розвитку галузі ці онтології продовжуватимуть відігравати вирішальну роль у розвитку технологій охорони здоров'я та покращенні догляду за пацієнтами.

У таблиці 2.1 представлені онтології в медицині.

Таблиця 2.1 – Онтології в медицині

Назва онтології	Сфера зосередження	Основні характеристики	Виробник
SNOMED CT [33]	Клінічна Термінологія	Повний перелік медичних термінів Підтримка різних мов. Детальна класифікація хвороб, симптомів, процедур	SNOMED International
МКХ (Міжнародна класифікація хвороб) [34]	Класифікація Хвороб	Визнана на світовому рівні Категоризація хвороб та станів здоров'я. Регулярно оновлюється	ВООЗ (Світова організація здоров'я)
LOINC (Logical Observation Identifiers Names and Codes) [35]	Лабораторні та Клінічні Спостереження	Стандарт для лабораторних тестів та клінічних спостережень Сприяє обміну та пулінгу результатів. Широко використовується в лабораторній звітності	Regenstrief Institute
Генна Онтологія (GO) [36]	Генетика та Молекулярна Біологія	Категоризація функцій генів. Структурована навколо біологічних процесів, клітинних компонентів, молекулярних функцій. Широко використовується в генетичних дослідженнях.	Gene Ontology Consortium
Основна Модель Анатомії (FMA) [37]	Анатомія Людини	Детальне представлення анатомічних структур. Ієрархічна організація. Інтегрується з іншими онтологіями для всебічного аналізу.	Університет Вашингтона

2.3 Інтерфейси програмування для онтологій

Було запропоновано багато інструментів для інтерфейсу RDF-графів або онтологій OWL, і багато з них перераховані на вебсайті W3C2. Однак існуючі інструменти мало допомагають у вирішенні двох труднощів, які ми визначили в попередньому розділі. Кілька інструментів пропонують міркування в закритому світі, але варіанти більш обмежені для локального закритого світу. Більшість інструментів зосереджено на RDF, деякі з них підтримують OWL, але вони рідко надають спеціальний синтаксис для полегшення використання класів і конструкцій заповнення ролей.

Ми можемо виділити три основні стратегії доступу до онтології на мові програмування (рис. 2.1). Перша стратегія полягає у використанні мови запитів, такої як SPARQL (протокол SPARQL і мова запитів RDF) [20]. SPARQL – це мова запитів, запропонована W3C. Дозволяє шукати та змінювати RDF-графіки. Його надихнув SQL (Structured Query Language), розроблений для реляційних баз даних. Однак SPARQL має два важливих недоліки. По-перше, це не об'єктно-орієнтований підхід: запит повинен бути написаний для кожного доступу до онтології, навіть найпростішого (наприклад, отримати значення функціональної властивості *r* для окремого *x*). Це може бути виснажливим, особливо якщо кількість різних запитів велика.

По-друге, SPARQL базується на RDF, а не на OWL. Таким чином, SPARQL є орієнтованим на дані, він виконує запити, але не робить висновки. Крім того, можна запитувати окремих осіб в онтології, але маніпулювання класами, наприклад додавання нових обмежень, є більш складним, хоча все ще можливим [21]. Цей момент є особливо проблематичним у біомедичній сфері, оскільки ми бачили, що більшість медичних концепцій, таких як розлади та медикаментозне лікування, є класами в онтологіях.

Query language (SPARQL) :

```
PREFIX <file://path/to/drug.owl>
SELECT (SUM(?drug_price) AS ?total_price)
WHERE {
    ?order :name "My order" .
    ?order :has_drug ?drug .
    ?drug :has_price ?drug_price .
}
```

Application Programming Interface (Java + OWLAPI) :

```
OWLDataFactory df = OWLManager.getOWLDataFactory();
public static float getOrderCost(OWLIndividual order) {
    float cost = 0.0;
    OWLObjectProperty drugProperty = df.getOWLObjectProperty(IRI.create("file.owl#has_drug"));
    OWLDataProperty priceProperty = df.getOWLDataProperty(IRI.create("file.owl#has_price"));
    Iterator drugs = order.listPropertyValues(drugProperty);
    while(drugs.hasNext()) {
        OWLIndividual drug = (OWLIndividual) drugs.next();
        Float price = (Float) drug.getPropertyValue(priceProperty);
        cost = cost + price.floatValue();
    }
    return cost;
}
```

Ontology-oriented programming (Python + Owlready) :

```
class Order(Thing):
    def get_cost(self):
        cost = 0.0
        for drug in self.drugs: cost = cost + drug.price
        return cost
```

Рисунок 2.1 – Приклад застосування інтерфейсів онтологій

Друга стратегія полягає у використанні інтерфейсу прикладного програмування (API). Ці API надають об'єкти та функції для маніпулювання елементами, які складають онтологію (тобто класи, індивідууми, властивості, анотації, обмеження тощо). У цій стратегії клас в онтології є екземпляром класу `OWLClass` в об'єктно-орієнтованій мові програмування. Існує кілька подібних API, наприклад `OWLAPI` [22] на Java або `owlcpp` [23] для C++ із прив'язкою для інших мов (включаючи Python). Ці API широко використовуються, але вони зосереджені на продуктивності, а не на простоті використання. Вони призводять до складних і об'ємних вихідних кодів (див. приклад на рисунку 2.1). Крім того, оскільки клас `OWL` розглядається як екземпляр у мові програмування, а не як клас, неможливо повністю скористатися перевагами об'єктно-орієнтованого програмування. У прикладі рис1, обчислення суми реалізовано у статичній функції, тоді як, щоб суворо дотримуватися парадигми об'єкта, воно мало бути реалізоване в методі класу `Order`. Однак API не може асоціювати методи з класами `OWL`.

Третя стратегія – онтологічно-орієнтоване програмування. Він заснований на об'єктно-орієнтованому програмуванні, добре відомій і успішній парадигмі. Він використовує переваги подібності, яка існує між онтологіями та моделями об'єктів [24]: класи, властивості та індивіди в онтології відповідають класам, атрибутам і екземплярам в об'єктній моделі [25].

Онтологічно-орієнтоване програмування намагається з'єднати або навіть уніфікувати онтології та об'єктну модель даної мови програмування, наприклад, клас в онтології стає класом у мові програмування. W3C показав, що цей підхід може зменшити довжину вихідного коду вдвічі [25]. На рисунку 2.1, онтологічно орієнтоване програмування не тільки зменшує довжину вихідного коду, але також суворіше дотримується об'єктної парадигми, визначаючи метод `get_cost()` у класі `Order` онтології. Можна виділити два підходи до онтологічно-орієнтованого програмування. Статичний підхід автоматично генерує вихідні коди для сутностей, присутніх в онтології (тобто класів, властивостей тощо). Він покладається на програму «перекладач», яка читає опис онтології (зазвичай файл OWL) і створює один або більше вихідних файлів, що містять класи на даній мові програмування. Статичний підхід полегшує доступ до вмісту онтології та виконує перевірку типу під час компіляції. Навпаки, через свою статичність цей підхід не може виконувати висновки або автоматичну класифікацію. Кілька статичних підходів були запропоновані для Java [26], Python[27] і C# [28].

З іншого боку, динамічний підхід складається з динамічного «перекладу» між онтологією та об'єктною моделлю під час виконання. Цей підхід дозволяє отримати доступ до онтології, але також може запустити механізм міркування та оновити об'єктну модель відповідно до зроблених висновків. Наприклад, автоматична класифікація може динамічно змінювати класи, до яких належить індивід. Крім того, цей підхід також дозволяє інтегрувати методи в класи онтології. Однак він не може виконувати перевірку типів, оскільки типи можуть змінюватися під час виконання [38].

Динамічний інструмент був запропонований у Common Lisp [24], [29],

використовуючи спеціальний алгоритм підрахунку, а не зовнішній аргумент. Прототип у Python також був створений за допомогою метакласів [30]; однак міркування та вираження обмежень залишаються досить обмеженими.

Зовсім недавно в Java був запропонований напівдинамічний підхід [32]. Він може виконувати автоматичну класифікацію індивідів за допомогою резонера з деякими обмеженнями (неможливо змінити клас екземпляра в статичній мові програмування, такій як Java, тому нові екземпляри створюються після класифікації, і програміст більше не повинен використовувати старші) [39].

При роботі з онтологіями в розробці програмного забезпечення різні інтерфейси та мови пропонують різні сильні та слабкі сторони. Проведено порівняння деяких популярних варіантів: SPARQL, Java, C, Python і Lisp (табл. 2.2).

Таблиця 2.2 – Порівняння програмних інтерфейсів для роботи із онтологіями

Мова програмування	Призначення	Переваги	Недоліки
1	2	3	4
SPARQL	Запити та маніпулювання даними RDF	Розроблено спеціально для запитів до онтологій. Добре інтегрується з базами даних RDF. Пропонує багатий набір можливостей запитів для складних структур даних.	Обмежений поза сферою RDF і онтологічних запитів. Може бути багатослівним для складних запитів.
Java	Використання: програмування загального призначення з бібліотеками для маніпулювання онтологіями (наприклад, Jena та OWL API).	Сильні об'єктно-орієнтовані можливості роблять його придатним для великомасштабних програм. Багата екосистема бібліотек для роботи з онтологіями. Хороша продуктивність і масштабованість.	Більш шаблонний код порівняно з мовами сценаріїв. Крута крива навчання для початківців в управлінні онтологіями.

Продовження таблиці 2.2

1	2	3	4
C	Високопродуктивне програмування на системному рівні [40].	Забезпечує високий контроль і ефективність, підходить для критично важливих додатків. Може використовуватися для побудови базових систем для баз даних онтології.	Не підходить безпосередньо для маніпулювання онтологіями; не має абстракцій високого рівня. Вимагає більше зусиль для керування пам'яттю та запобігання помилкам.
Python	Основне використання: програмування загального призначення з такими бібліотеками, як RDFLib і Додаток2 для роботи з онтологіями [41].	Простий у вивченні та використанні, з чітким і читабельним синтаксисом. Багаті бібліотеки та підтримка спільноти для управління онтологіями. Добре підходить для швидкої розробки та створення прототипів.	Повільніший час виконання порівняно з скомпільованими мовами, такими як Java і C. Динамічний тип може викликати помилки, які не були виявлені під час компіляції.
Lisp	Штучний інтелект і дослідження, з можливостями для символічних міркувань [42].	Чудово підходить для символічних обчислень, що є корисним для онтологічних міркувань. Дуже гнучка та адаптивна мова, добре підходить для експериментальної роботи.	Менш відомі та використовувані, що призводить до меншої кількості спільнот і меншої кількості бібліотек. Унікальний синтаксис і парадигми можуть бути складними для тих, хто звик до основних мов.

Вибір значною мірою залежить від конкретних вимог проєкту, таких як потреба в продуктивності, простота розробки або спеціалізовані можливості маніпулювання онтологіями.

2.4 Порівняння об'єктних моделей і формальних онтологій

Об'єктні моделі та формальні онтології схожі в багатьох аспектах. Існує декілька об'єктних моделей (фактично по одній на кожен об'єктно-орієнтовану мову програмування), і можна виділити дві основні категорії: статичні об'єктні моделі (наприклад, у Java), у яких класи компілюються та статично типізуються під час компіляції, та динамічні об'єктні моделі (наприклад, у Python), які динамічно вводяться під час виконання. Динамічні об'єктні моделі є загалом більш дозволеними, наприклад, клас екземпляра можна змінити під час виконання. Нові атрибути також можна додати до екземпляра, навіть якщо ці атрибути не були оголошені на рівні класу. Таблиця 2.3 показує відмінності між статичними об'єктними моделями, динамічними об'єктними моделями та формальними онтологіями. Очевидно, що динамічні об'єктні моделі ближчі до онтологій, ніж статичні.

Таблиця 2.3 – Порівняння статичних об'єктних моделей, динамічних об'єктних моделей та формальних онтологій

Характеристика	Статичні об'єктні моделі (напр. Java)	Динамічні об'єктні моделі (напр. Python)	Формальні онтології (напр. OWL)
1	2	3	4
Припущення закритого світу	Так	Так	Ні (припущення відкритого світу)
Неявне розділення класів	Так (класи роз'єднані, якщо жоден з них не є дочірнім до іншого)	Так (класи роз'єднані, якщо у них немає спільного нащадка)	Ні (розділення класів має бути вказане явно)
Розрізнення між екземплярами	Так (всі екземпляри різні)	Так (всі екземпляри різні)	Ні (розрізнення між індивідами має бути вказане явно)
Спадкування	Одиночне	Множинне	Множинне
Інстанціювання	Одиночне	Одиночне	Множинне

Продовження таблиці 2.3

1	2	3	4
Зміна класу екземпляра в рантаймі	Неможливо	Можливо	Можливо (класи можуть бути змінені на більш конкретні класи за допомогою механізму висновків)
Зміна надкласу класу в рантаймі	Неможливо	Можливо	Можливо (надкласи можуть бути змінені на більш конкретні надкласи за допомогою механізму висновків)
Атрибути	Визначені для даного класу	Визначені для даного класу, але можуть розглядатися як незалежні від класу (duck-typing)	Властивості незалежні від класів (вони є конструкціями першого порядку з підтримкою спадкування)
Підтримка анотацій	Немає	Немає	Є

Основні відмінності між моделями динамічних об'єктів і формальними онтологіями полягають у тому, чи вважаються невикладені факти помилковими або можливими (припущення закритого чи відкритого світу, відповідно). Чи можуть два класи бути еквівалентними та роз'єднаність класів. необхідно чітко вказати. Чи можуть два екземпляри/індивідууми бути однаковими та той факт, що вони різні, має бути зазначено явно. Чи підтримується множинна інстанціація. Чи атрибути/властивості є першорядними конструкції, незалежні від класів і підтримки успадкування, або визначені всередині даного класу. Чи підтримуються анотації.

Підсумовуючи, динамічні моделі об'єктів ближчі до онтологій, ніж

статичні. За розвиток нашої орієнтованої на онтологію модуль програмування, ми обрали Python, тому що це динамічна об'єктно-орієнтована мова програмування, тому що вона проста у використанні та широко використовується в біомедичній сфері, а також тому, що ми вже розробили кілька інструментів цією мовою, включаючи загальний API для медичної термінології [42].

2.5 Висновки до розділу 2

У другому розділі досліджено методи автоматичної класифікації медичних протипоказань. Проаналізовано можливості застосування онтологій, інтегрованих з машинним навчанням, для структурування та аналізу складних медичних даних щодо протипоказань. Розглянуто особливості медичних онтологій та напрямки їх розвитку.

Запропоновано використання онтологічно-орієнтованого підходу до програмування та інтеграції з мовою Python для спрощення доступу та маніпулювання онтологічними сутностями.

Отримані результати дозволяють перейти до наступного етапу - програмної реалізації методів автоматичної класифікації медичних протипоказань на основі розроблених теоретичних рішень.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДУ

3.1 Структурна схема програмної реалізації методів

Загальна структура додатку включає три компоненти (рис. 3.1):

- метакласи для класів і конструкцій OWL;
- необов'язкові специфічні для онтології вихідні файли Python, визначення методів для вставки в класи OWL;
- двигун SPARQL.

Додаток відповідає за імпорт онтологій у сховище, динамічне завантаження їхніх сутностей і загортання в об'єкти Python, імпорт вихідних файлів Python, пов'язаних із онтологіями, і виконання порівняння. Додаток представляє програмісту уніфікований інтерфейс, змішуючи оператори OWL (властивості, ієрархію класів, особи) з операторами Python (визначення методів).

Додаток використовує відкладений синтаксичний аналізатор для динамічного завантаження сутностей із сховища на вимогу: коли сутність (клас, властивість або окрема особа) отримує доступ у Python, вона завантажується в пам'ять, загортається в об'єкт Python, зберігається в кеш і повертається. Якщо об'єкт Python-оболонку змінено, сховище автоматично оновлюється шляхом додавання, видалення або зміни трійок RDF. Коли на оболонку більше не посилаються, а кеш заповнений, вона знищується. Потім сутність можна знову завантажити з RDF, якщо програмі потрібно знову отримати до неї доступ.

Визначені користувачем методи Python також можна асоціювати з класами OWL; у цьому випадку можна використовувати спеціальну анотацію ("python_module"), щоб зв'язати онтологію з назвою пов'язаного модуля Python. Нарешті, аналізатор можна використовувати для виконання автоматичної класифікації класів та осіб.

Ця архітектура підтримує великі онтології, що зберігаються в оптимізованому сховищі. Це забезпечує швидкий доступ для «переглядання» сутностей в онтологіях.

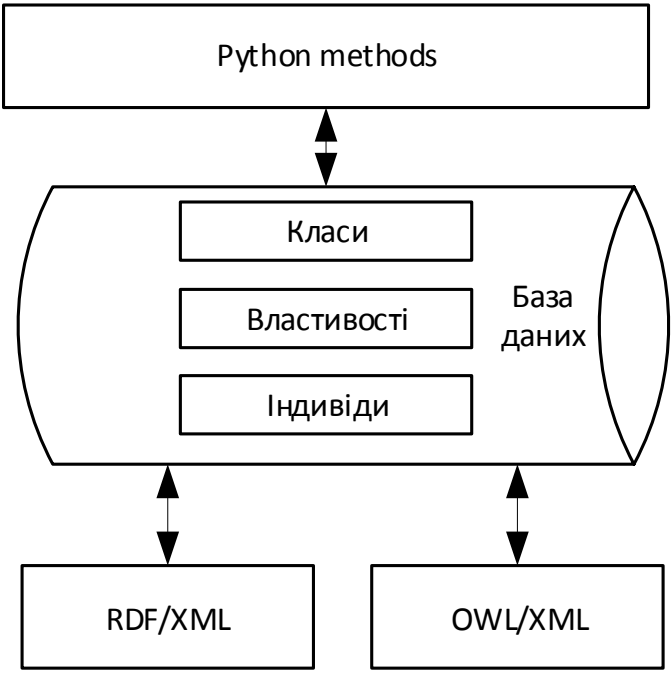


Рисунок 3.1 – Структура додатку

В таблиці 3.1 описані етапи роботи даного проєкту.

Таблиця 3.1 – Етапи роботи сервісу

Етап	Опис	Ключові дії
Імпорт онтологій в сховище	Завантаження онтологій OWL у сховище даних. Це включає читання файлів онтологій з зовнішніх джерел і перенесення їх у формат, сумісний із сховищем.	Читання файлів онтологій. Перенесення у сумісний формат
Динамічне завантаження сутностей	Використання відкладеного синтаксичного аналізатора для запиту сутностей з сховища. Сутності завантажуються в пам'ять, коли до них здійснюється запит у Python.	Запит сутностей за потребою. Динамічне завантаження в пам'ять.

Продовження таблиці 3.1

Етап	Опис	Ключові дії
Створення об'єктів Python	Завантажені сутності згортаються в об'єкти Python і зберігаються в кеші для підвищення продуктивності.	Згортання сутностей у Python-об'єкти. Зберігання в кеші.
Синхронізація змін з Сховищем	Оновлення сховища при зміні Python-оболонки: додавання, видалення або зміна RDF-трійок.	Оновлення сховища при змінах
Інтеграція специфічних для онтології Python-файлів	Імпорт Python-файлів, які визначають методи для класів OWL, і використання анотацій для зв'язування методів з онтологією.	Імпорт Python-файлів. Використання анотацій для зв'язування
Використання SPARQL для запитів та аналізу	Запити до онтологій за допомогою двигуна SPARQL для класифікації класів та осіб та отримання складних запитів інформації.	Використання SPARQL для запитів та аналізу
Обробка та видалення об'єктів	Управління пам'яттю шляхом знищення об'єктів, на які більше не посилаються, і коли кеш заповнений.	Видалення не використовуваних об'єктів.
Повторне завантаження за необхідності	Повторне завантаження сутностей з RDF, якщо потрібно знову отримати до них доступ.	Завантаження сутностей за потребою

3.2 Модель OWL на Python

Моделлю Python було отримати максимально прозорий доступ до онтологій у Python з точки зору програміста. Програміст повинен мати можливість маніпулювати класами та окремими особами онтології так, якби вони були звичайними класами та об'єктами Python. Додаток може виражати майже всі конструкції OWL 2.0, включаючи класи, індивідууми, властивості об'єктів, властивості даних, анотації, домени та діапазони властивостей, обмежені типи даних, роз'єднані, вирази класів, такі як перетини, об'єднання, обмеження значень властивостей.

Реалізація в Python спирається на метакласи. Клас OWL представлений у Python екземпляром метакласу, тобто класом Python. Таблиця 3.2 показує різні спеціальні методи, які були перевизначені для адаптації онтологій OWL до об'єктної моделі Python. Ці спеціальні методи визначаються або на рівні метакласу (тобто вони застосовуються до класів), або на рівні класу (тобто вони застосовуються до екземплярів).

Таблиця 3.2 – Функції інтерпертації онтологій OWL до об'єктної моделі Python

Спеціальний метод	Функція	Перевизначення
1	2	3
ClassCreation	Створити новий клас (тобто новий екземпляр метакласу).	Поєднати новий клас з уже існуючим класом OWL з такою ж назвою, якщо такий є.
ICheck	Перевірка, чи є даний об'єкт екземпляром цього класу.	Врахувати еквівалентні класи OWL.
SubjectChek	Перевірка, чи є даний клас підкласом цього класу.	Врахувати еквівалентні класи OWL.

Продовження таблиці 3.2

1	2	3
MROComputation	Обчислити порядок розв'язання методів (MRO), який використовується у випадку множинного успадкування.	Якщо Python не може знайти послідовний MRO, створити відповідний. Це відбувається у випадку складних ієрархій з множинним успадкуванням.
AttributeSetter	Встановити значення даного атрибуту класу.	Встановити значення властивості анотації та оновити квадростор. Визначити конструкцію ролі-заповнювача як властивість класу.
AttributeGetter	Отримати значення даного атрибуту класу	Запитати існуючі анотації у квадросторі, якщо такі є. Запитати конструкцію ролі-заповнювача як властивість класу
InstanceAttributeSetter	Встановити значення даного атрибуту цього екземпляра.	Встановити значення анотацій або властивостей та оновити квадростор. Оновити інверсну властивість, якщо така є.
InstanceAttributeGetter	Отримати значення даного атрибуту цього екземпляра (викликається лише для атрибутів, які ще не визначені).	Запитати існуючі анотації або відносини у квадросторі та кешувати результат.

OWL надає різні типи даних, які представлені RDF-літералами. Додаток підтримує такі типи даних: логічний, цілий, з плаваючою точкою, рядок, нормалізований рядок, локалізований рядок, дата, час і дата-час. Нормалізований рядок – це рядок без розриву рядка. Локалізований рядок – це літерал RDF певною мовою. Літерали зберігаються в сховище як рядок, який є конкатенація двох елементів: серіалізація значення та буквено-цифровий код, що ідентифікує тип даних IRI..

Онтології – це, по суті, структурований спосіб представлення знань, який часто використовується в штучному інтелекті, технологіях семантичної мережі та інформаційних системах.

В основі будь-якої онтології лежать класи (або поняття), які представляють набори сутностей або речей. Наприклад, у медичній онтології класами можуть бути «Хвороба» або «Симптом». Ці класи не просто мітки; вони визначають суть того, чим є ці сутності в контексті онтології.

Властивості (або відношення) є ще одним фундаментальним типом об'єкта. Вони описують різні способи, якими екземпляри класів можуть співвідноситись один з одним. Наприклад, властивість може з'єднати клас «Хвороба» з класом «Симптом», вказуючи, які симптоми пов'язані з якими захворюваннями. Ці властивості можна далі класифікувати на властивості об'єктів, які пов'язують екземпляр одного класу з екземпляром іншого, і властивості типу даних, які пов'язують екземпляри класу зі значеннями даних.

Індивіди (або екземпляри) – це фактичні сутності, які представляють класи. У медичній онтології окрема особа може бути певною хворобою, як-от «діабет». Ці особи є конкретними прикладами абстрактних понять, визначених класами.

Нарешті, аксіоми - це твердження або твердження, які завжди істинні в контексті онтології. Вони використовуються для моделювання необхідних умов для різних класів і властивостей. Наприклад, аксіома може стверджувати, що всі екземпляри класу «Симптом» мають бути пов'язані принаймні з одним екземпляром класу «Хвороба».

Разом ці об'єкти працюють у тандемі, щоб створити багату та взаємопов'язану модель певної області знань. Ця модель не тільки дозволяє організувати та категоризувати інформацію, але також підтримує міркування про зв'язки та властивості сутностей у цьому домені.

Створення онтології медичних протипоказань вимагає уваги до кількох ключових предметних областей та їх структур:

- фармацевтичні препарати. Описувати властивості ліків, такі як активні інгредієнти, дозування, форма випуску (таблетки, розчини, ін'єкції тощо). Також важливо зазначити взаємодію між різними препаратами (рис. 3.2);
- медичні стани та захворювання. Описувати різні стани та захворювання, включаючи їх симптоми, етіологію, патофізіологію, а також взаємодію із лікарськими засобами;
- алергії та індивідуальна непереносимість. Важливо зазначити інформацію про алергії та індивідуальну непереносимість пацієнта до певних речовин;
- лабораторні показники. Включити дані про лабораторні показники, які можуть бути важливими для визначення протипоказань (наприклад, рівень креатиніну для оцінки функції нирок) (рис. 3.3).
- клінічні процедури та хірургічні втручання. Інформація про нещодавні або майбутні клінічні процедури та хірургічні втручання, які можуть впливати на лікування (рис. 3.4).
- інші медичні препарати та добавки. Урахування взаємодії з вітамінами, травами, добавками, немедичними препаратами.

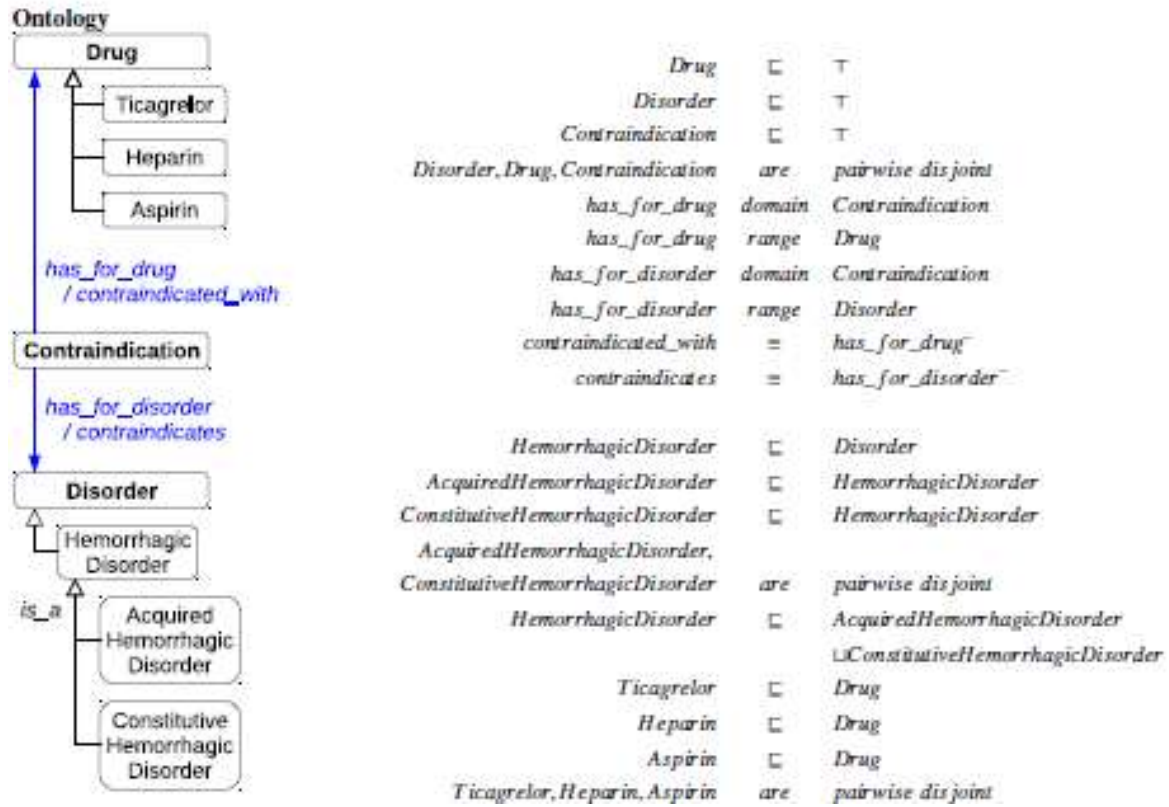


Рисунок 3.2 – Приклад структури онтології протипоказань

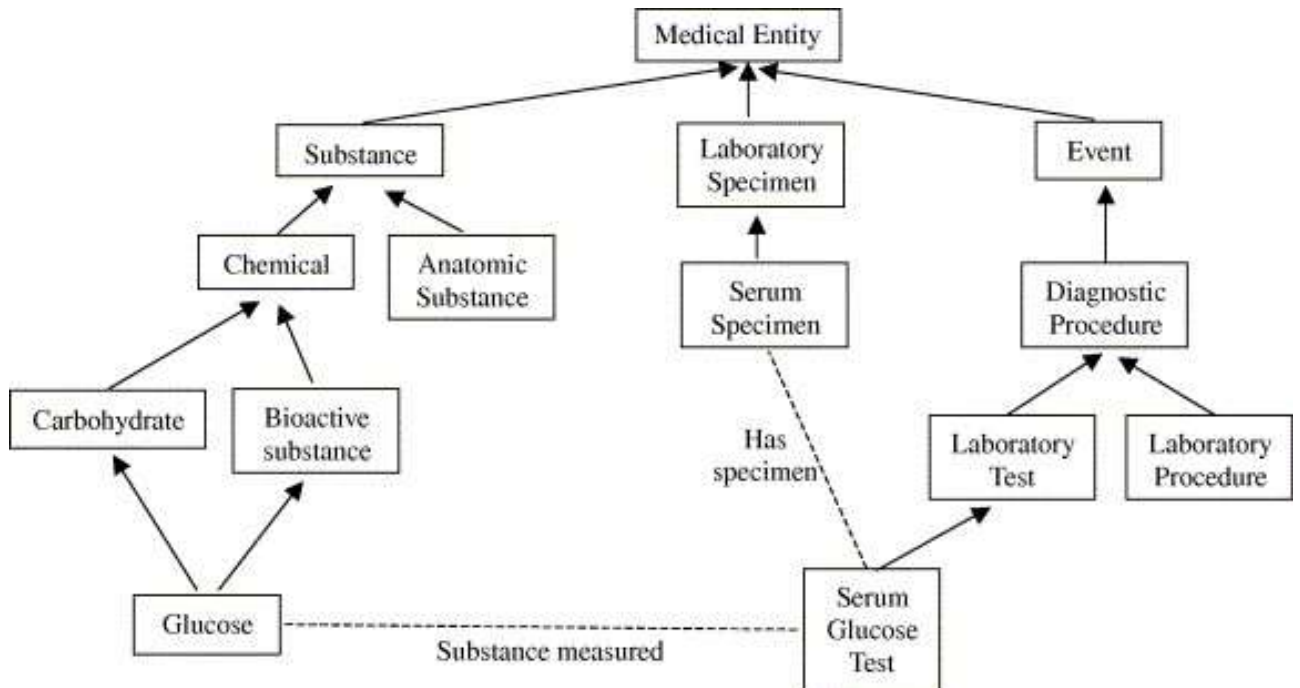


Рисунок 3.3 – Онтологія класів аналізів

На рисунку 3.4 наведена онтологія лікування.

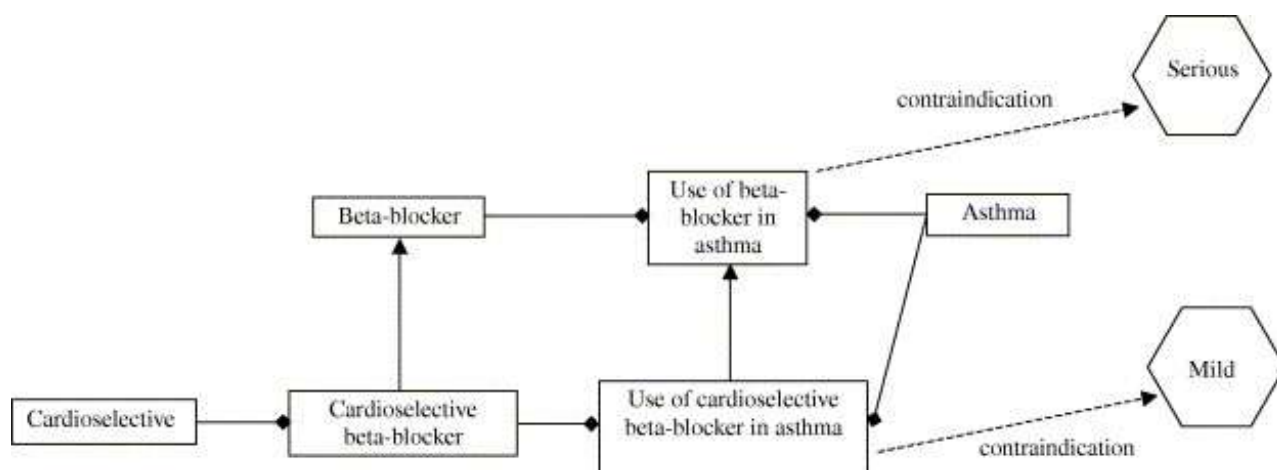


Рисунок 3.4 – Онтологія лікування

3.3 Функції доступу до онтології

Нотаціями редактора Protégé, так і крапковою нотацією, яка широко використовується в об'єктно-орієнтованому програмуванні, включаючи Python.

Багато об'єктно-орієнтованих мов програмування, включаючи Python, покладаються на нотацію з крапками. Щоб імітувати об'єктно-орієнтоване програмування, Додаток використовує крапкову нотацію на п'яти рівнях:

- для доступу до сутностей, визначених в онтології. Вважається, що кожна онтологія визначає свій власний простір імен, і крапкова нотація використовується для доступу до сутностей у цьому просторі імен. Наприклад, якщо гіпотетична онтологія ([URIhttp://www.test.org/drug.owl](http://www.test.org/drug.owl)) завантажується в змінну `drug_onto`, клас `Drug`, визначений у цій онтології ([URIhttp://test.org/drug.owl#Drug](http://test.org/drug.owl#Drug)) можна отримати доступ як `drug_onto.Drug`. Надаються додаткові методи для отримання списку класів, властивостей, осіб або анотацій в онтології тощо;

- отримати доступ або змінити вартість власності для особи. Наприклад, щоб отримати значення властивості «hasIngredient» для особи в змінній «test_drug», можна написати «test_drug.hasIngredient». Функціональні властивості повертають одне значення, за замовчуванням – «Немає» (нульове значення Python), якщо не визначено. Нефункціональні властивості повертають

список значень використовується для додавання елемента до списку. Нарешті, значення зворотних властивостей оновлюються автоматично. В онтологіях властивості часто розглядаються як відношення і, таким чином, називаються дієсловом, наприклад `hasX` замість `X`, але це не зазвичай у програмуванні мови тому додаток пропонує анотацію «pythonDiya», яку можна використовувати для вказівки іншої назви при використанні крапкової нотації. Наприклад, після визначення трійки анотацій (`hasIngredient`, `pythonDiya`, `Class`), тепер можна отримати доступ до значень властивості більш звичайним способом;

- щоб отримати доступ або змінити анотації для будь-якої сутності (класу, власності чи окремої особи). Це працює подібно до властивостей, але з властивостями анотації;

- щоб отримати доступ або змінити конструкції, пов'язані з класом. З об'єктно-орієнтованої точки зору ці конструкції заповнювача ролей можна розглядати як атрибути класу;

- отримати доступ або змінити домен, діапазон і властивості.

Додаток підтримує три формати файлів онтології: NTriples, RDF/XML і OWL/XML, використовуючи прості парсери, написані на Python. Під час завантаження онтології додаток автоматично рекурсивно завантажує всі імпортовані онтології.

Щоб полегшити доступ до локальних копій онтологій, ми також визначили глобальну змінну. Вона містить список каталогів, у яких здійснюється пошук локальних копій.

Помилки, виявлені під час завантаження онтології, повідомляються шляхом створення винятку. Додаток використовує декілька методів для завантаження онтологій залежно від формату файлу. Для всіх методів ми дотримувалися тієї самої схеми звітування про помилки та використовували той самий виняток у разі виявлення помилки. Крім того, якщо деякі RDF-трійки вже були завантажені, коли сталася помилка, вони видаляються зі сховища. Отже, невдале завантаження залишає квадрокопію без змін. Якщо онтологію неможливо знайти, виникає виняток `HTTPError`. Оскільки кожна онтологія має

IRI, очікується, що він буде доступний онлайн. Таким чином, відсутність онтології призводить до помилки HTTP.

Сховище може зберігатися в пам'яті (поведінка за замовчуванням) або на диску (у файлі). Коли він зберігається на диску, він забезпечує постійність: усі попередньо завантажені онтології відновлюються під час повторного використання того самого файлу. Ця функція цікава для великих онтологій, оскільки відкриття бази даних з диска займає лише частки секунди.

Нарешті, додаток може експортувати онтології у форматах NTriples і RDF/XML, OWL/XML.

3.4 Висновки до розділу 3

У третьому розділі описано програмну реалізацію методу автоматичної класифікації медичних протипоказань на основі онтологій та мови програмування Python. Розроблено архітектуру програмного додатку, що включає інтерфейс доступу до онтології, механізми завантаження та маніпулювання онтологічними сутностями, функції для виконання міркувань.

Запропоновано способи спрощення роботи з онтологіями в Python за рахунок онтологічно-орієнтованого підходу та спеціальних функцій.

Отримані результати дають можливість перевірити ефективність розробленого методу автоматичної класифікації медичних протипоказань за допомогою програмної реалізації.

4 ТЕСТУВАННЯ МЕТОДІВ АВТОМАТИЧНОЇ КЛАСИФІКАЦІЇ МЕДИЧНИХ ПРОТИПОКАЗАНЬ

4.1 Автоматична класифікація та міркування

Додаток майже не робить висновків, доки аргумент не буде виконано явно. Ця поведінка подібна до поведінки редактора Protégé, у якому користувач повинен явно викликати аргументатор, натиснувши на меню. Єдиним винятком є значення інверсних властивостей, які автоматично оновлюються.

Резонер виконується викликом глобальної функції `sync_reasoner()`. Ця функція експортує сховище RDF у тимчасовий файл у форматі NTriples. Потім він запускає аргумент `HermiT` для цього файлу. Ми використовували `HermiT` версії 5.3.8 і модифікували її, щоб генерувати на стандартному виводі класифікацію класів, властивостей та індивідів у вигляді звичайного тексту (`HermiT` не пропонує класифікацію індивідів у параметрах командного рядка, хоча аргументація виконує класифікацію). Функція `sync_reasoner()` отримує та аналізує вихідні дані, створені `HermiT`. Нарешті, він оновлює онтології в Додаток за допомогою виведених операторів, виведених `HermiT`. Ці твердження є новими відношеннями `є`, `є екземпляром`, еквівалентно і те саме як.

Оновлення включає два етапи:

- оновлення RDF-сховища к шляхом додавання RDF-трійок для виведених операторів;
- оновлення об'єктів Python-оболонки, якщо вони були завантажені та все ще доступні в кеші. Наприклад, якщо `HermiT` вивів це `AB`, функція `sync_reasoner()` додасть трійку RDF (`A`, `subclassOf`, `B`) у сховище. Крім того, якщо клас `A` було завантажено в Python, суперкласи класу Python `A` будуть змінені, щоб включити клас `B`. Якщо клас `A` ще не завантажено, але завантажуватиметься після виконання аргументу, новий (`A`, `subclassOf`, `B`) RDF triple буде враховано під час завантаження класу, і, таким чином, клас `A` включатиме `B` у свої суперкласи, як і очікувалося.

Виведені факти можна перевірити в Python за допомогою стандартних

методів самоаналізу об'єктів. Наприклад, стандартну функцію Python `issubclass(A, B)` можна використовувати для перевірки того, чи є клас `A` підкласом `B`. Додаток також може зберегти виведені факти в новій онтології.

Ми побачили, що для медичних міркувань часто потрібні локальні міркування в закритій предметній області. Додаток покладається на розроблені онтології та правила для міркувань. Щоб дозволити локальне міркування закритої предметної області, сервіс надає функцію `close_world()`. Він очікує єдиного аргументу, який є індивідом або класом. Ця функція діє як «препроцесор», який генерує відповідні універсальні обмеження, необхідні для розгляду даної особи чи класу в закритому світі. Для індивіда це означає, що індивідуальні відносини обмежені стверджуваними відносинами та екзистенціальними обмеженнями. Для класу це означає, що всі індивідууми класу відомі, і що всі підкласи та індивідууми даного класу також розглядаються в закритому світі.

Обмеження, створені `close_world()`, залежать від заявлених відносин (для індивідів), обмежень (для індивідів і класів) та індивідів (для класів). Для індивіда, для кожної властивості, він розглядає як заявлені значення, так і екзистенціальні обмеження, і створює відповідне універсальне обмеження. Для класів функція виконує 3 кроки. По-перше, якщо в класі є індивідууми, це створює обмеження "один із" (інші розумно, ми припускаємо, що особи не були стверджені, хоча вони можуть існувати, єдиною альтернативою є розглядати клас як еквівалент `Nothing`). По-друге, для кожної властивості створюється універсальне обмеження, залежне від визначених обмежень для класу, його предків, нащадків та їхніх осіб. По-третє, він рекурсивно звертається до підкласів та індивідів.

Викликаючи `close_world()` для обмеженого набору індивідів або класів, можна міркувати в локальному закритому світі.

Якщо потрібне перемикання назад до міркувань відкритого світу, найкращим варіантом є встановлення обмежень, створених `close_world()` у новій онтології (використовуючи оператор «with...», як описано вище). Потім

видалення цієї онтології з Quadstore усуває обмеження, повертаючись до міркування відкритого світу.

Спочатку ми побудували онтологію протипоказань. Наркотики і наркотикотерапевтичні класи були витягнуті з французької бази даних лікарських засобів Thériaque. Захворювання та протипоказання були витягнуті з SPC та закодовані вручну експертом-фармацевтом, щоб уникнути помилок у базі даних. Спадкові зв'язки між розладами були отримані з медичної термінології ICD10 (Міжнародна класифікація хвороб, випуск 10). Розділи були визначені за допомогою методів NLP (обробка природної мови). Наркотиків і розладів бути не може представлені індивідами, і їх потрібно моделювати як класи, щоб дозволити відносини між ними. Ми також визначили клас протипоказань, пов'язаний як з ліками, так і з розладами.

Для кожного препарату протипоказання були витягнуті вручну з довідкових документів. Потім вони були підтверджені в онтології за допомогою додатку за п'ять кроків:

- створено індивідуальне для кожного протипоказання;
- пов'язан кожен препарат з його протипоказаннями;
- пов'язане кожне захворювання з його протипоказаннями;
- ствердження що про протипоказання відомо все.

Крок 1 полягає у створенні індивідуумів і не представляє труднощів.

Етапи 2 і 3 полягають у створенні обмежень «заповнення ролей» для класів (ліки та розлади) та екзистенціальних обмежень для індивідів (протипоказання). Цього можна легко досягти за допомогою крапкової нотації із синтаксисом, запропонованим нами для «заповнювача ролей»: «Class.property = values».

Етапи 4 і 5 полягають у тому, щоб стверджувати, що про протипоказання та ліки відомо все (тобто аргументатор не повинен будувати гіпотези щодо невідомих протипоказань). Цього можна легко досягти, викликавши функцію `close_world()` для класів `Contraindication` і `Drug` (які рекурсивно закривають усі екземпляри `Contraindication` і всі підкласи `Drug`).

Навпаки, клас Disorder має залишатися у відкритому світі, якщо ми хочемо вивести протипоказання за допомогою міркувань (за допомогою субсумпції та розділів). Тому міркування відбуваються в локальному закритому світі.

Підсумовуючи, цей варіант використання ілюструє інтерес синтаксису високого рівня, запропонованого Додаток для міркування локального замкнутої предметної області та для визначення конструкцій заповнювача ролей та екзистенційних обмежень, а також його здатність мати справу з класами в простому вигляді. способом, включаючи автоматичну класифікацію індивідів і класів. На рисунку 4.1, програма Python явно коротша та легша для читання та розуміння, ніж відповідні твердження в нотації DL.

Python program	Corresponding assertions in formal notation
(1) <code>ci1 = Contraindication()</code> <code>ci2 = Contraindication()</code> <code>ci3 = Contraindication()</code> <code>ci4 = Contraindication()</code>	$Contraindication(ci1)$ $Contraindication(ci2)$ $Contraindication(ci3)$ $Contraindication(ci4)$
(2) <code>Ticagrelor.contraindicated_with = [ci1]</code> <code>Heparin.contraindicated_with = [ci2]</code> <code>Aspirin.contraindicated_with = [ci3, ci4]</code>	$Ticagrelor \sqsubseteq \exists \text{contraindicated_with}.\{ci1\}$ $(\exists \text{has_for_drug.Ticagrelor})(ci1)$ $Heparin \sqsubseteq \exists \text{contraindicated_with}.\{ci2\}$ $(\exists \text{has_for_drug.Heparin})(ci2)$ $Aspirin \sqsubseteq \exists \text{contraindicated_with}.\{ci3\}$ $(\exists \text{has_for_drug.Aspirin})(ci3)$ $Aspirin \sqsubseteq \exists \text{contraindicated_with}.\{ci4\}$ $(\exists \text{has_for_drug.Aspirin})(ci4)$
(3) <code>HemorrhagicDisorder.contraindicates = [ci1]</code> <code>AcquiredHemorrhagicDisorder.contraindicates = [ci3]</code> <code>ConstitutiveHemorrhagicDisorder.contraindicates = [ci2, ci4]</code>	$HemorrhagicDisorder \sqsubseteq \exists \text{contraindicates}.\{ci1\}$ $(\exists \text{has_for_disorder.HemorrhagicDisorder})(ci1)$ $AcquiredHemorrhagicDisorder \sqsubseteq \exists \text{contraindicates}.\{ci3\}$ $(\exists \text{has_for_disorder.AcquiredHemorrhagicDisorder})(ci3)$ $ConstitutiveHemorrhagicDisorder \sqsubseteq \exists \text{contraindicates}.\{ci2\}$ $(\exists \text{has_for_disorder.ConstitutiveHemorrhagicDisorder})(ci2)$ $ConstitutiveHemorrhagicDisorder \sqsubseteq \exists \text{contraindicates}.\{ci4\}$ $(\exists \text{has_for_disorder.ConstitutiveHemorrhagicDisorder})(ci4)$
(4) <code>close_world(Contraindication)</code>	$(\forall \text{has_for_drug.Ticagrelor})(ci1)$ $(\forall \text{has_for_disorder.HemorrhagicDisorder})(ci1)$ $(\forall \text{has_for_drug.Heparin})(ci2)$ $(\forall \text{has_for_disorder.ConstitutiveHemorrhagicDisorder})(ci2)$ $(\forall \text{has_for_drug.Aspirin})(ci3)$ $(\forall \text{has_for_disorder.AcquiredHemorrhagicDisorder})(ci3)$ $(\forall \text{has_for_drug.Aspirin})(ci4)$ $(\forall \text{has_for_disorder.ConstitutiveHemorrhagicDisorder})(ci4)$ $Contraindication \sqsubseteq \{ci1, ci2, ci3, ci4\}$
(5) <code>close_world(Drug)</code>	$Ticagrelor \sqsubseteq \forall \text{contraindicated_with}.\{ci1\}$ $Heparin \sqsubseteq \forall \text{contraindicated_with}.\{ci2\}$ $Aspirin \sqsubseteq \forall \text{contraindicated_with}.\{ci3, ci4\}$

Рисунок 4.1 – Програма Python і відповідні твердження в нотації DL

Досконаліша програма може повною мірою використати можливості,

надані Python, для запитів до баз даних ліків або створення графічних інтерфейсів для введення даних (рис. 4.2).

Рисунок 4.2 – Інтерфейс для вводу препаратів

Є можливість використовувати сутності в онтології так, ніби вони були класами та об'єктами Python. Це дозволяє повторно використовувати існуючі бібліотеки Python з окремими особами в онтології замість об'єктів Python. Крім того, завдяки синтаксису «заповнювачі ролей як атрибути класу», класи онтології також можна використовувати як об'єкти Python.

Можливо, синтаксис додатку більш стислий, ніж синтаксис OWLAPI дозволяє коротший вихідний код. OWLAPI дозволяє маніпулювати кожною аксіомою онтології окремо; Хоча це може бути цікаво в деяких конкретних ситуаціях, часто це не є вимогою. Навпаки, запропонований онтологічно-орієнтований синтаксис програмування є коротшим і простішим у використанні та задовольняє потреби більшості програм.

Використовуючи додаток, просте визначення онтології (без складних логічних виразів, таких як екзистенціальні та універсальні обмеження) майже так само легко, як створення ієрархії класів у мові програмування. Таким чином, це може зробити будь-який розробник. Потім онтологія може додати обмеження у файл OWL, наприклад, використовуючи редактор Protégé. Нарешті, додаток здатний поєднувати об'єктну модель з онтологією в одній

моделі.

Додаток також можна використовувати як «клей» для легкого доступу до онтології в Python, а потім для підключення її до інших компонентів (наприклад, вебсайту). Він використовувався студентами магістратури (M1 з біомедичної інформатики), які майже не мали базових знань в онтології та DL, забезпечуючи їм легкий та об'єктно-орієнтований доступ до сутностей в онтології.

4.2 Експериментальне дослідження

Для класифікації розладів щодо аспірину можна використовувати два наступних класи.

Подібні класи можна визначити для інших препаратів. Після запуску резонера розлади будуть автоматично класифіковані відповідно до цих визначених класів. Усі розлади, які є протипоказанням до аспірину, будуть зазначені як підкласи DisorderContraindicatin-gAspirin, усі розлади, для яких можна довести відсутність протипоказань до аспірину, будуть заявлятися як підкласи DisorderOkWithAspirin («OK» у таблиці), і всі розлади, які протипоказані в деяких ситуаціях, але не всі, не є підкласами жодного з цих двох класів («CI/Ok» у таблиці). Автоматичну класифікацію можна виконати в Python, а потім стандартну функцію `issubclass()` можна використати для перевірки, чи є клас підкласом іншого, наприклад, `issubclass(HemorrhagicDisorder, DisorderOkWithAspirin)` поверне значення False.

Додаток Б включає комп'ютерний код “contraindications.txt” для прикладу варіанту використання.

Приклад, який ми описали, був простим і включав лише три препарати, три розлади та чотири протипоказання. Однак на практиці препарати набагато складніші. Таблиця 4.1 показує показники для чотирьох наборів даних, над якими ми працювали в проєкті. Кожен набір даних відповідає новому препарату, який нещодавно вийшов на ринок, а також включає кілька

препаратів порівняння (тобто вже існуючі препарати з тим самим показанням). У таблиці також вказано кількість висновків, отриманих для кожного набору даних.

Таблиця 4.1 – Набір даних

Приклад	Кількість ліків	Кількість розладів	Кількість заявлених протипоказань	Кількість розділів	Кількість розділених (за розладами)	Кількість висновків: протипоказання
Аспірин	8	29	70	9	640	23
Парацетомол	4	17	30	4	166	23
Ібупрофен	9	14	42	1	82	7
Но-шпа	5	26	50	8	499	-

На рисунку 4.3 представлено експериментальне дослідження.

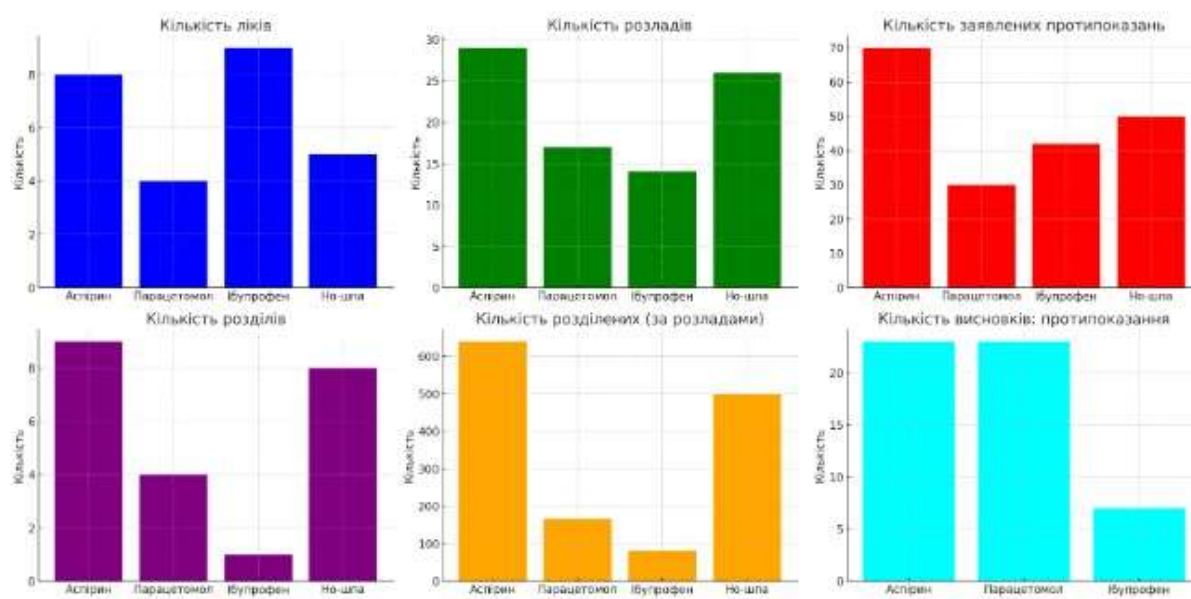


Рисунок 4.3 – Експериментальне дослідження

Висновки до класифікації ліків:

- кількість ліків. Ібупрофен має найбільшу кількість лікарських форм серед чотирьох препаратів, тоді як Парацетамол має найменшу;
- кількість розладів. Аспірин асоціюється з найбільшою кількістю

розладів, на які він може впливати, в той час як Ібупрофен має відносно меншу кількість розладів;

- кількість заявлених протипоказань. Аспірин має найбільшу кількість заявлених протипоказань, що може вказувати на його широкий спектр впливу та необхідність обережності при його використанні. Парацетамол має найменшу кількість заявлених протипоказань;

- кількість розділів.: Аспірин та Но-шпа мають велику кількість розділів, що може свідчити про різноманітність їхніх застосувань та побічних ефектів. Натомість Ібупрофен має лише один розділ;

- кількість розділених (за розладами). Аспірин має найбільшу кількість розділених (за розладами), що може вказувати на його складність в лікуванні різних станів. Но-шпа також показує високі показники у цій категорії;

- кількість висновків протипоказання: І Аспірин, і Парацетамол мають однакову кількість висновків щодо протипоказань, що може свідчити про їх важливість у медичній практиці. Для Но-шпи дані відсутні.

4.3 Висновки за розділом

У четвертому розділі проведено тестування розробленого методу автоматичної класифікації медичних протипоказань та його програмної реалізації за допомогою онтологій. Експериментально перевірено можливості автоматичної класифікації медичних станів та протипоказань на основі логічних міркувань та використання онтологій. Продемонстровано функції класифікації даних за допомогою розробленого програмного додатку.

Отримані результати свідчать про ефективність розробленого підходу та його здатність вирішувати задачі автоматичної класифікації протипоказань на основі складних онтологічних даних і міркувань. Розроблений програмний додаток може бути покладений в основу інтелектуальних систем підтримки прийняття медичних рішень.

ВИСНОВКИ

Під час кваліфікаційної роботи магістра було вирішено актуальну науково-прикладну задачу підвищення ефективності автоматичної класифікації медичних протипоказань на основі розробки та програмної реалізації удосконалених методів з використанням інтеграції онтологій та машинного навчання.

Результати дослідження дозволяють створити інтелектуальну систему підтримки прийняття рішень з урахуванням персональних даних пацієнта та протипоказань до лікування. Розроблене програмне забезпечення може бути використане для реалізації такої системи в медичних закладах.

Основними науковими результатами є удосконалення методу автоматичної класифікації протипоказань та розробка програмного додатку на основі мови Python, що інтегрує онтології та машинне навчання.

Практична значимість отриманих результатів полягає в можливості їх застосування для підвищення якості медичного обслуговування та запобігання помилкам на етапі призначення лікування. Подальші дослідження будуть спрямовані на розширення функціональних можливостей розробленого програмного забезпечення та його впровадження на практиці.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Бабінцева Л. Ю. Онтології в фармації [Електрон ресурс]. / Л. Ю. Бабінцева // Медична інформатика та інженерія. – 2015. – № 3. – Режим доступу: <https://doi.org/10.11603/mie.1996-1960.2014.3.3798>.
2. Ляшкевич М. Проектування онтології розмітки для систем керування контентом [Електрон. ресурс] / Марія Ляшкевич, Василь Ляшкевич, Роман Шувар // Information technology: computer science, software engineering and cyber security. – 2023. – № 1. – С. 67–73. – Режим доступу: <https://doi.org/10.32782/it/2023-1-9>.
3. Даревич Р. Р. Автоматизована метапошукова система на основі адаптивної онтології : автореф. дис. ... канд. техн. наук / Даревич Р. Р. ; Нац. ун-т "Львівська політехніка". – Львів, 2007. – 20 с.
4. Stryzhak O. Y. Комп'ютерні онтології – технологічна основа формування освітянських інформаційних ресурсів [Електрон. ресурс]. / Oleksandr Ye Stryzhak, Valentyna B. Demyanenko // Information technologies and learning tools. – 2011. – Т. 22, № 2. – Режим доступу: <https://doi.org/10.33407/itlt.v22i2.419>.
5. Mintser O. P. Про загальне впорядкування медико-біологічних знань. створення онтології [Електрон. ресурс]. / O. P. Mintser, L. Yu Babintseva // Medical informatics and engineering. – 2015. – № 2. – Режим доступу: <https://doi.org/10.11603/mie.1996-1960.2014.2.3802>.
6. Чаплінський Ю. П. Представлення прийняття рішень в багаторівневих системах за допомоги онтології [Електрон. ресурс]. / Ю. П. Чаплінський, О. В. Субботіна // Problems of informatization and management. – 2009. – Т. 3, № 27. – Режим доступу: <https://doi.org/10.18372/2073-4751.3.567>.
7. Аналіз онтології як моделі представлення знань текстової інформації [Електрон. ресурс]. : thesis / Вавіленкова Анастасія Ігорівна [та ін.]. – [Б. м.], 2010. – Режим доступу: <http://er.nau.edu.ua/handle/NAU/14888>.

8. Дацюк О. Інструментальний засіб підключення реляційних баз даних до онтології предметної області [Електрон. ресурс]. / Оксана Дацюк, Ілля Шестеріков // Міжгалузеві диспути: динаміка та розвиток сучасних наукових досліджень. – [Б. м.], 2020. – Режим доступу: <https://doi.org/10.36074/10.07.2020.v1.09>.
9. Jean-Baptiste L. OWL ontologies [Electronic resource]. / Lamy Jean-Baptiste // Ontologies with Python. – Berkeley, CA, 2020. – P. 49–82. – Mode of access: https://doi.org/10.1007/978-1-4842-6552-9_3.
10. Jean-Baptiste L. Mixing Python and OWL [Electronic resource]. / Lamy Jean-Baptiste // Ontologies with Python. – Berkeley, CA, 2020. – P. 241–264. – Access mode: https://doi.org/10.1007/978-1-4842-6552-9_10.
11. Bertaud-Gounot V. Ontology and medical diagnosis [Electronic resource]. / Valérie Bertaud-Gounot, Régis Duvauferrier, Anita Burgun // Informatics for Health and Social Care. – 2011. – Vol. 37, no. 2. – P. 51–61. – Access mode: <https://doi.org/10.3109/17538157.2011.590258>.
12. Sadegh-Zadeh K. Medical Ontology [Electronic resource]. / Kazem Sadegh-Zadeh // Philosophy and Medicine. – Dordrecht, 2011. – P. 711–756. – Access mode: https://doi.org/10.1007/978-94-007-2260-6_19.
13. Kim H. G. Ontology Technology in Medical Informatics [Electronic resource]. / Hong Gee Kim, Myeng Ki Kim // Journal of Korean Society of Medical Informatics. – 2003. – Vol. 9, no. 3. – P. 213. – Access mode: <https://doi.org/10.4258/jksmi.2003.9.3.213>.
14. Elçi A. Ontology-Based Smart Medical Solutions [Electronic resource]. / Atilla Elçi, Duygu Çelik Ertuğrul // Expert Systems. – 2020. – Vol. 37, no. 1. – Access mode: <https://doi.org/10.1111/exsy.12518>.
15. Reflections on a medical ontology [Electronic resource]. / Bo Hu [et al.] // International Journal of Human-Computer Studies. – 2007. – Vol. 65, no. 7. – P. 569–582. – Access mode: <https://doi.org/10.1016/j.ijhcs.2007.02.005>.
16. Gene Ontology Consortium. The Gene Ontology (GO) database and informatics resource [Electronic resource]. / Gene Ontology Consortium // Nucleic

Acids Research. – 2004. – Vol. 32, no. 90001. – P. 258D–261. – Access mode: <https://doi.org/10.1093/nar/gkh036>.

17. Lee Y. Ontology integration: Experience with medical terminologies [Electronic resource]. / Yugyung Lee, Kaustubh Supekar, James Geller // Computers in Biology and Medicine. – 2006. – Vol. 36, no. 7-8. – P. 893–919. – Access mode: <https://doi.org/10.1016/j.combiomed.2005.04.013>.

18. Integration of medical ontology concepts to annotate medical images [Electronic resource]. / Mohd Nizam Saad [et al.] // Jurnal teknologi. – 2015. – Vol. 77, no. 29. – Access mode: <https://doi.org/10.11113/jt.v77.6807>.

19. Lebedev S. V. Ontology-driven approach to medical data fusion [Electronic resource]. / S. V. Lebedev, N. A. Zhukova // Ontology of designing. – 2017. – Vol. 7, no. 2. – P. 145–159. – Access mode: <https://doi.org/10.18287/2223-9537-2017-7-2-145-159>.

20. Concept extraction for medical documents using ontology [Electronic resource]. // International journal of advance engineering and research development. – 2016. – Vol. 3, no. 03. – Access mode: <https://doi.org/10.21090/ijaerd.0303124>.

21. Alrumaih H. Developing a Medical Ontology for Diabetes Treatment [Electronic resource] / Hala Alrumaih, Mohammed Alawairdhi // Journal of Computational and Theoretical Nanoscience. – 2019. – Vol. 16, no. 9. – P. 3783–3791. – Access mode: <https://doi.org/10.1166/jctn.2019.8251>.

22. Parker M. The body multiple: Ontology in medical practice [Electronic resource]. / Michael Parker // BioSocieties. – 2007. – Vol. 2, no. 2. – P. 281–282. – Access mode: <https://doi.org/10.1017/s174585520722558x>.

23. Zeshan F. Medical Ontology in the Dynamic Healthcare Environment [Electronic resource]. / Furkh Zeshan, Radziah Mohamad // Procedia Computer Science. – 2012. – Vol. 10. – P. 340–348. – Access mode: <https://doi.org/10.1016/j.procs.2012.06.045>.

24. Moon K.-S. Oriental Medical Ontology for Personalized Diagnostic Services [Electronic resource]. / Kyung-Sil Moon, Su-Hyun Park // Journal of the Korea Society of Computer and Information. – 2010. – Vol. 15, no. 1. – P. 23–30. –

Access mode: <https://doi.org/10.9708/jksci.2010.15.1.023>.

25. Mol A. Body Multiple: Ontology in Medical Practice / Annemarie Mol, Barbara Herrnstein Smith, E. Roy Weintraub. – [S. l.] : Duke University Press, 2002.

26. Modeling a medical environment: an ontology for integrated medical informatics design [Electronic resource]. / John R. Hajdukiewicz [et al.] // International Journal of Medical Informatics. – 2001. – Vol. 62, no. 1. – P. 79–99. – Access mode: [https://doi.org/10.1016/s1386-5056\(01\)00128-9](https://doi.org/10.1016/s1386-5056(01)00128-9).

27. Medical diagnosis ontology for intelligent decision support systems [Electronic resource]. / V. V. Gribova [et al.] // Ontology of designing. – 2018. – Vol. 8, no. 1. – P. 58–73. – Access mode: <https://doi.org/10.18287/2223-9537-2018-8-1-58-73>.

28. Gorskis H. Ontology-based system development for medical database access [Electronic resource]. / Henrihs Gorskis, Ludmila Aleksejeva, Inese Polaka // Environment. technology. resources. proceedings of the international scientific and practical conference. – 2017. – Vol. 2. – P. 24. – Access mode: <https://doi.org/10.17770/etr2017vol2.2572>.

29. Duffin J. Review: the body multiple: ontology in medical practice [Electronic resource]. / J. Duffin // Journal of the History of Medicine and Allied Sciences. – 2004. – Vol. 59, no. 2. – P. 319–321. – Access mode: <https://doi.org/10.1093/jhmas/jrh084>.

30. Ontology-based Knowledge Representation for Supporting Medical Decisions [Electronic resource]. / José A. Reyes-Ortiz [et al.] // Research in Computing Science. – 2013. – Vol. 68, no. 1. – P. 127–136. – Access mode: <https://doi.org/10.13053/rcs-68-1-11>.

31. Wennerberg P. Ontology modularization to improve semantic medical image annotation [Electronic resource]. / Pinar Wennerberg, Klaus Schulz, Paul Buitelaar // Journal of Biomedical Informatics. – 2011. – Vol. 44, no. 1. – P. 155–162. – Access mode: <https://doi.org/10.1016/j.jbi.2010.12.005>.

32. Ontology-Based Generation of Medical, Multi-term MCQs [Electronic resource]. / J. Leo [et al.] // International Journal of Artificial Intelligence in

Education. – 2019. – Vol. 29, no. 2. – P. 145–188. – Access mode: <https://doi.org/10.1007/s40593-018-00172-w>.

33. Na J. Effectiveness of UMLS semantic network as a seed ontology for building a medical domain ontology [Electronic resource]. / Jin-Cheon Na, Hock Leng Neoh // *Aslib Proceedings*. – 2008. – Vol. 60, no. 1. – P. 32–46. – Access mode: <https://doi.org/10.1108/00012530810847353>.

34. Yi M. Effective medical resources searching using an ontology-driven medical information retrieval system [Electronic resource]. / Myongho Yi // *The Electronic Library*. – 2012. – Vol. 30, no. 2. – P. 248–263. – Access mode: <https://doi.org/10.1108/02640471211221368>.

35. Leroy G. Meeting Medical Terminology Needs - the ontology-enhanced medical concept mapper [Electronic resource]. / Leroy Gony, Chen Hsinchun. – [S. l.], 2001. – Access mode: <http://hdl.handle.net/10150/105242>.

36. Kim G.-J. Development of Ontology based Medical Mobile CRM (m-CRM) [Electronic resource]. / Gui-Jung Kim, Jung-Soo Han // *Journal of the Korea Academia-Industrial cooperation Society*. – 2009. – Vol. 10, no. 10. – P. 2721–2727. – Access mode: <https://doi.org/10.5762/kais.2009.10.10.2721>.

37. Medical document clustering using ontology-based term similarity measures [Electronic resource]. / Xiaodan Zhang [et al.] // *International journal of data warehousing and mining*. – 2008. – Vol. 4, no. 1. – P. 62–73. – Access mode: <https://doi.org/10.4018/jdwm.2008010104>.

38. Ladychuk O. K. Ontology as a means of presentation of medical information [Electronic resource]. / O. K. Ladychuk // *Scientific notes of junior academy of sciences of ukraine*. – 2021. – No. 21-22. – Access mode: https://doi.org/10.51707/2618-0529-2021-21_22-09.

39. Ontology-Based Categorisation of Medical Texts for Health Professionals [Electronic resource]. / Antonio Balderas [et al.] // *Proceedings*. – 2018. – Vol. 2, no. 19. – P. 1203. – Access mode: <https://doi.org/10.3390/proceedings2191203>.

40. Akremi H. To medical ontology fuzzification purpose: COVID-19 study

case [Electronic resource]. / Houda Akremi, Mouhamed Gaith Ayadi, Sami Zghal // Procedia computer science. – 2022. – Vol. 207. – P. 1027–1036. – Access mode: <https://doi.org/10.1016/j.procs.2022.09.158>.

41. Jensen T. E. Book review: the body multiple: ontology in medical practice [Electronic resource]. / Torben Elgaard Jensen, Brit Ross Winthereik // Acta sociologica. – 2005. – Vol. 48, no. 3. – P. 266–268. – Access mode: <https://doi.org/10.1177/000169930504800309>.

42. Ontology-based approach to enhance medical web information extraction [Electronic resource]. / Nassim Abdeldjallal Otmani [et al.] // International Journal of Web Information Systems. – 2019. – Vol. 15, no. 3. – P. 359–382. – Access mode: <https://doi.org/10.1108/ijwis-03-2018-0017>.

ДОДАТОК А
Технічне завдання

Вступ

Програмне забезпечення може використовуватися для автоматичної класифікації медичних протипоказань.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Дослідження та програмна реалізація методів автоматичної класифікації медичних протипоказань», затверджене наказом Національного університету «Запорізька політехніка» № 437 від 14 листопада 2023 р.

A.2 Призначення розробки

Програмний продукт призначений для автоматичної класифікації медичних протипоказань на основі даних про лікарський препарат.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

При розробці програмного забезпечення необхідно забезпечити такі функціональні вимоги:

- імпорт даних про ліки та протипоказання;
- автоматична класифікація протипоказань на основі даних;
- візуалізація результатів класифікації.

A.3.2 Вимоги до інтерфейсу програми

Інтерфейс користувача має бути інтуїтивно зрозумілим та зручним для роботи медичних працівників.

A.3.3 Вимоги до надійності

Програмне забезпечення для автоматичної класифікації медичних протипоказань має коректно обробляти дані та уникати помилок класифікації.

A.3.4 Умови експлуатації

Програмне забезпечення призначене для використання у медичних закладах. Для експлуатації потрібна наявність персонального комп'ютера, доступ до медичних даних та баз знань.

A.3.5 Вимоги до складу та параметрів технічний засобів

Персональний комп'ютер. Процесор - Intel Core i5. Оперативна пам'ять - 8 Гб і більше.

A.3.6 Вимоги до маркування і пакування

Програмне забезпечення може бути записана на будь-якому носії інформації. На пакуванні повинна бути назва програми – "Система автоматичної класифікації медичних протипоказань".

ДОДАТОК Б**Онтологія медичної предметної області**

```
<?xml version="1.0"?>
```

```
<!DOCTYPE rdf:RDF [
  <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
  <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
  <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
  <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
]>
```

```
<rdf:RDF
  xmlns="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#"

  xml:base="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <owl:Ontology
    rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL"/>
```

```
<!--
```

```
////////////////////////////////////
////////////////////////////////////
////////////////////////////////////
//
// Classes
//

////////////////////////////////////
////////////////////////////////////
-->
```

```

<!-- http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#A -->

<owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#A">
    <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Вітаміни"/>
    </owl:Class>

<!-- http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#B1 --
>

<owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#B1"
>
    <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Вітаміни"/>
    </owl:Class>

<!-- http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#B12 -
->

<owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#B12
">
    <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Вітаміни"/>
    </owl:Class>

<!-- http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#B2 --
>

```

```

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#B2"
>
    <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Вітаміни"/>
    </owl:Class>

```

```

<!-- http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#B5 --
>

```

```

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#B5"
>
    <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Вітаміни"/>
    </owl:Class>

```

```

<!-- http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#B6 --
>

```

```

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#B6"
>
    <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Вітаміни"/>
    </owl:Class>

```

```

<!-- http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#B9 --
>

```

```

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#B9"
>
    <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Вітаміни"/>
    </owl:Class>

```

```

<!-- http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#C -->

```

```

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#C">
    <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Вітаміни"/>
    </owl:Class>

```

```

<!-- http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#D -->

```

```

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#D">
    <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Вітаміни"/>
    </owl:Class>

```

```

<!-- http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#E -->

```

```

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#E">
    <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#

```

```

Вітаміни"/>
    </owl:Class>

    <!-- http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#K -->

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#K">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Вітаміни"/>
    </owl:Class>

    <!-- http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#PP --
>

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#PP"
>
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Вітаміни"/>
    </owl:Class>

    <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#dissectio -->

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#dis
sectio">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Хірургічні_втручання"/>
    </owl:Class>

```

```

    <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#incisio -->

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#inc
isio">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Хірургічні_втручання"/>
    </owl:Class>

    <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#paphia -->

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#pap
hia">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Хірургічні_втручання"/>
    </owl:Class>

    <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#punctio -->

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#pun
ctio">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Хірургічні_втручання"/>
    </owl:Class>

```

```

    <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#resectio -->

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#res
ectio">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Хірургічні_втручання"/>
    </owl:Class>

    <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#scopia -->

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#sco
pia">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Хірургічні_втручання"/>
    </owl:Class>

    <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#sectio -->

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#sec
tio">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Хірургічні_втручання"/>
    </owl:Class>

```

```

    <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#stomio -->

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#stomio">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Хірургічні_втручання"/>
    </owl:Class>

    <!-- http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#tonia
-->

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#tonia">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Хірургічні_втручання"/>
    </owl:Class>

    <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Ібупрофен -->

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Ібупрофен">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Препарати"/>
    </owl:Class>

```

```

    <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Алепрії -->

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Але
прії"/>

    <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Аспірин -->

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Асп
ірин">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Препарати"/>
    </owl:Class>

    <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Барітність -->

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Бар
ітність">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Протипоказання"/>
    </owl:Class>

    <!--

```

```
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Важка_ниркова_
і_печінкова_недостатність -->
```

```
<owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Важ
ка_ниркова_і_печінкова_недостатність">
  <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Протипоказання"/>
</owl:Class>
```

```
<!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Воднорозчинні
-->
```

```
<owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Вод
норозчинні">
  <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Вітаміни"/>
</owl:Class>
```

```
<!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Вітаміни -->
```

```
<owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Віт
аміни"/>
```

```
<!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Добавки -->
```

```

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Добавки"/>

```

```

    <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Діагностувальні -->

```

```

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Діагностувальні">
    <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Хірургічні_втручання"/>
    </owl:Class>

```

```

    <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Діти_до_2_років -->

```

```

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Діти_до_2_років">
    <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Протипоказання"/>
    </owl:Class>

```

```

    <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Екстрені -->

```

```

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Екс

```

```

трени">
    <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Хірургічні_втручання"/>
    </owl:Class>

    <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Етапні -->

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Ета
пні">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Хірургічні_втручання"/>
        </owl:Class>

        <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Жирозчинні -->

        <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Жир
озчинні">
            <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Вітаміни"/>
            </owl:Class>

            <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Захворювання -
->

            <owl:Class

```

```

rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Зах
ворювання"/>

```

```

<!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Клінічні_проце
дури -->

```

```

<owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Клі
нічні_процедури"/>

```

```

<!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Лабораторні_по
казники -->

```

```

<owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Лаб
ораторні_показники"/>

```

```

<!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Лікувальні -->

```

```

<owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Лік
увальні">

```

```

    <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Хірургічні_втручання"/>

```

```

    </owl:Class>

```

```

<!--

```

```
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Медичні_стани
-->
```

```
<owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Мед
ичні_стани"/>
```

```
<!-- http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Хо-
шпа -->
```

```
<owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Хо-
шпа">
```

```
<rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Препарати"/>
```

```
</owl:Class>
```

```
<!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Одночасний_при
йом_інгібіторів_МАО -->
```

```
<owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Одн
очасний_прийом_інгібіторів_МАО">
```

```
<rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Протипоказання"/>
```

```
</owl:Class>
```

```
<!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Паліативні -->
```

```

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Паліативні">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Хірургічні_втручання"/>
    </owl:Class>

```

```

    <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Парацетомол --
>

```

```

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Парацетомол">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Препарати"/>
    </owl:Class>

```

```

    <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Первинна -->

```

```

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Первинна">
        <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Хірургічні_втручання"/>
    </owl:Class>

```

```

    <!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Планові -->

```

```

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Планові">
    <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Хірургічні_втручання"/>
    </owl:Class>

```

```

<!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Повторні -->

```

```

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Повторні">
    <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Хірургічні_втручання"/>
    </owl:Class>

```

```

<!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Порушення_кровообігу -->

```

```

    <owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Порушення_кровообігу">
    <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Протипоказання"/>
    </owl:Class>

```

```

<!--

```

```
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Препарати -->
```

```

    <owl:Class
      rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Препарати"/>

```

```

    <!--
      http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Протипоказання
    -->

```

```

    <owl:Class
      rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Протипоказання"/>

```

```

    <!--
      http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Радикальні -->

```

```

    <owl:Class
      rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Радикальні">
      <rdfs:subClassOf
        rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Хірургічні_втручання"/>
    </owl:Class>

```

```

    <!--
      http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Сильна_аналгетична_активність -->

```

```

    <owl:Class
      rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Сильна_аналгетична_активність">
      <rdfs:subClassOf

```

```

rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Протипоказання"/>
</owl:Class>

```

```

<!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Термінові -->

```

```

<owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Тер
мінові">
    <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Хірургічні_втручання"/>
    </owl:Class>

```

```

<!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Хірургічні_втр
учання -->

```

```

<owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Хіп
ургічні_втручання"/>

```

```

<!--
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#бронхіальна_ас
тма -->

```

```

<owl:Class
rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#бро
нхіальна_астма">
    <rdfs:subClassOf
rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#
Протипоказання"/>

```

```
</owl:Class>
```

```
<!--
```

```
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#виразкова_хвороба -->
```

```
<owl:Class
```

```
  rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#виразкова_хвороба">
```

```
    <rdfs:subClassOf
```

```
      rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Протипоказання"/>
```

```
  </owl:Class>
```

```
<!--
```

```
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#лактація -->
```

```
<owl:Class
```

```
  rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#лактація">
```

```
    <rdfs:subClassOf
```

```
      rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#Протипоказання"/>
```

```
  </owl:Class>
```

```
<!--
```

```
http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#непереносимість_хворим_конкретного_препарату -->
```

```
<owl:Class
```

```
  rdf:about="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#непереносимість_хворим_конкретного_препарату">
```

```
    <rdfs:subClassOf
```

```
    rdf:resource="http://www.semanticweb.org/admin/ontologies/2023/11/medOWL#  
Протипоказання"/>  
  </owl:Class>  
</rdf:RDF>
```

```
<!--      Generated      by      the      OWL      API      (version      3.4.2)  
http://owlapi.sourceforge.net -->
```

ДОДАТОК В
Слайди презентації

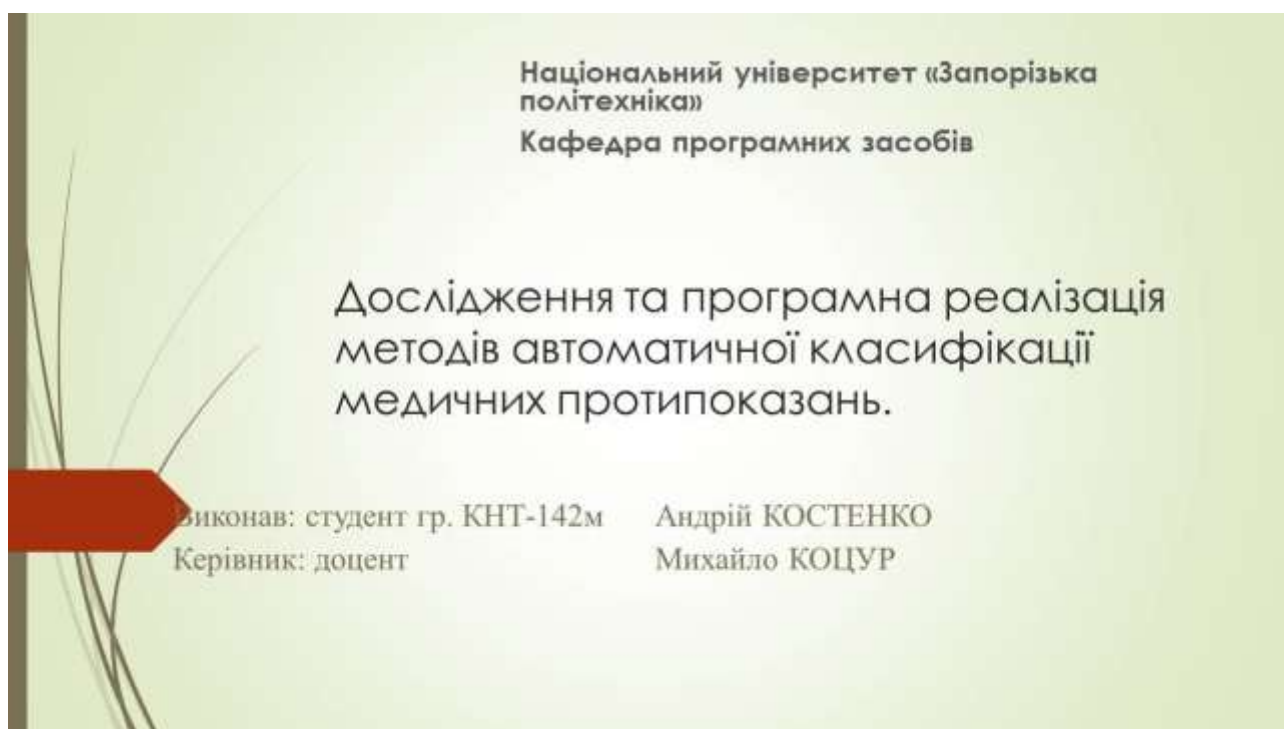


Рисунок В.1 – Слайд 1



Рисунок В.2 – Слайд 2

3

Зберігання даних на програмному рівні для медичних протипоказань

Параметр	Опис
Дизайн бази даних	Програмне забезпечення має використовувати надійну базу даних для зберігання даних протипоказань. Ця база даних включатиме таблиці для ліків, медичних умов, демографічних даних пацієнтів і особливостей протипоказань.
Нормалізація даних	Забезпечення узгодженості даних шляхом нормалізації бази даних для зменшення надмірності та залежності.
Безпека та відповідність	Впровадження надійних протоколів безпеки та дотримання правил охорони здоров'я, таких як HIPAA, для захисту конфіденційної інформації пацієнтів.

Рисунок В.3 – Слайд 3

4

Аналіз онтологій в медицині

Назва Онтології	Сфера Зосередження	Основні Характеристики	Виробник
SNOMED CT	Клінічна Термінологія	Повний перелік медичних термінів Підтримка різномов Детальна класифікація хвороб, симптомів, процедур	SNOMED International
МКХ (Міжнародна класифікація хвороб)	Класифікація Хвороб	Визнана на світовому рівні Категоризація хвороб та станів здоров'я Регулярно оновлюється	WHO (Світова організація здоров'я)
LOINC (Logical Observation Identifiers Names and Codes)	Лабораторні та Клінічні Спостереження	Стандарт для лабораторних тестів та клінічних спостережень Сприяє обміну та пулінгу результатів Широко використовується в лабораторній звітності	Regenstrief Institute
Генна Онтологія (GO)	Генетика та Молекулярна Біологія	Категоризація функцій генів Структурована навколо біологічних процесів, клітинних компонентів, молекулярних функцій Широко використовується в генетичних дослідженнях	Gene Ontology Consortium
Основна Модель Анатомії (FMA)	Анатомія Людини	Детальне представлення анатомічних структур Ієрархічна організація Інтегрується з іншими онтологіями для всебічного аналізу	Університет Вашингтона

Рисунок В.4 – Слайд 4

5		Етапи дослідження	
Задача	Опис	Задача	Опис
Аналіз вимог	Розуміти типи протипоказань, які необхідно класифікувати.	Збір даних і керування ними:	Використовувати відкриті джерела
Вибір технологій	Провести аналіз технологій автоматичної обробки даних.	Розробка онтології	Використання онтологій для структурування медичних даних, що дозволяє краще класифікувати та розуміти складні медичні зв'язки.
Розробка програмного забезпечення	Розробка програми з акцентом на зручність користування інтерфейсами.	Тестування та перевірка	Щоб забезпечити точність і надійність.

Рисунок В.5 – Слайд 5

6		Порівняння програмних інтерфейсів для роботи із онтологіями	
Назва програми/застосунку	Призначення	Переваги	Недоліки
SPARQL	застосунок для маніпуляційних запитів RDB	Розроблено спеціально для запитів до онтологій, добре інтегровано з базами даних RDB. Пропонує багатий набір можливостей запитів для кожного типу даних.	Обмежений коло сфери RDB і онтологічних запитів. Може бути болючим для складних запитів.
Java	використання програмувальних запитів, призначених з бібліотеками для придатності великомасштабних програм, маніпулювання онтологіями (наприклад, Java Biomedical Ontology API або OWL API).	Сильні об'єктно-орієнтовані можливості роботи з даними. Значно швидший код порівняно з мовою C. Хороша продуктивність/масштабованість.	Занадто багато шаблонного коду порівняно з мовою C. Криво, криво написані для пошуку в управлінні онтологіями.
C	високопродуктивне програмувальне середовище для системного рівня.	Надійшли високі показники ефективності, підходить для критично важливих запитів. Може використовуватися для побудови базових систем для роботи з базами даних онтологій.	Не підходить безпосередньо для маніпулювання онтологіями, немає об'єктної моделі роботи з даними. Більше зусиль для керування пам'яттю та запобігання помилкам.
Python	Осередок використання програмувальних запитів, призначених з бібліотеками, як RDB, так і OWL API для роботи з онтологіями.	Простий у вивченні та використанні, з чітким і читабельним синтаксисом. Багато бібліотек та підтримка спільнот для управління онтологіями. Добре підходить для швидкої розробки та старання програм.	Повільний порівняно з мовою C. Динамічний тип може виконати помилку, як не були вивчені типи даних.
PHP	Швидкий інтерпретатор і дослідження, можливість для символічного коду.	Зручно підходить для символічного коду, що є корисним для онтологічних мов. Дуже легко та швидко можна добре підходить для експериментальної роботи.	Менш відомі та використовуються, що призводить до меншої кількості спільнот/меншої кількості бібліотек. Унікальний синтаксис і парадигми можуть бути складними для тих, хто звик до основних мов.

Рисунок В.6 – Слайд 6

7

Порівняння статичних об'єктних моделей, динамічних об'єктних моделей та формальних онтологій

Характеристика	Статичні об'єктні моделі (напр. Java)	Динамічні об'єктні моделі (напр. Python)	Формальні онтології (напр. OWL)
Припущення закритого світу	Так	Так	Ні (припущення відкритого світу)
Невизначені розділення класів	Так (класи роз'єднані, якщо жоден з них не є дочірнім до іншого)	Так (класи роз'єднані, якщо у них немає спільного нащадка)	Ні (розділення класів має бути вказано явно)
Розрізнення між екземплярами	Так (всі екземпляри різні)	Так (всі екземпляри різні)	Ні (розрізнення між індивідами має бути вказано явно)
Споживання ієрархії класів	Одиничне	Множинне	Множинне
Зміна класу екземпляра в рантаймі	Неможливо	Можливо	Можливо (класи можуть бути змінені на більш конкретні класи за допомогою механізму висновків)
Зміна надкласу класу в рантаймі	Неможливо	Можливо	Можливо (надкласи можуть бути змінені на більш конкретні надкласи за допомогою механізму висновків)
Атрибути	Визначені для даного класу	Визначені для даного класу, але можуть розкладатися як невизначені від класу (duck-typing)	Властивості незалежні від класів (вони є конструкціями першого порядку з підтримкою споживання)
Підтримка анотацій	Немає	Немає	Є

Рисунок В.7 – Слайд 7

8

Структура програмної реалізації методу

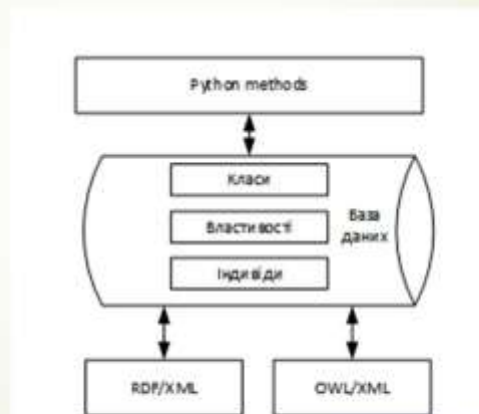


Рисунок В.8 – Слайд 8



Рисунок В.9 – Слайд 9



Рисунок В.10 – Слайд 10



Висновки

- Під час кваліфікаційної роботи магістра було вирішено актуальну науково-прикладну задачу підвищення ефективності автоматичної класифікації медичних протипоказань на основі розробки та програмної реалізації удосконалених методів з використанням інтеграції онтологій та машинного навчання.
- Результати дослідження дозволяють створити інтелектуальну систему підтримки прийняття рішень з урахуванням персональних даних пацієнта та протипоказань до лікування. Розроблене програмне забезпечення може бути використане для реалізації такої системи в медичних закладах.
- Основними науковими результатами є удосконалення методу автоматичної класифікації протипоказань та розробка програмного додатку на основі мови Python, що інтегрує онтології та машинне навчання.
- Практична значимість отриманих результатів полягає в можливості їх застосування для підвищення якості медичного обслуговування та запобігання помилкам на етапі призначення лікування. Подальші дослідження будуть спрямовані на розширення функціональних можливостей розробленого програмного забезпечення та його впровадження на практиці.

Рисунок В.11 – Слайд 11