Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра інформатики факультету інформатики

Проектування та реалізація корпоративної бази знань

Текстова частина до курсової роботи
за спеціальністю „Інженерія програмного забезпечення” 121

Керівник курсової роботи
доц. Жежерун О.П.
_____
(підпис)
“____” _____2020 р.

Виконала студентка ІПЗ-16
Манжура Анна
“____” _____2020 р.

Київ 2020

Календарний план виконання роботи:

№ п/п	Назва етапу дипломного проекту (роботи)	Термін виконання етапу	Примітка
1.	Отримання завдання на курсову роботу.	02.10.2019	
2.	Узгодження теми, структури та термінів виконання з керівником.	04.12.2019	
3.	Огляд технічної літератури за темою роботи.	30.01.2020	
4.	Ознайомлення з функціональністю та можливостями редактора Protege.	16.02.2020	
5.	Розробка бази знань медичного закладу.	01.03.2020	
6.	Виконання тестування спроектованої системи та написання звіту по практичній частині.	10.03.2020	
7.	Написання текстової частини курсової роботи.	13.03.2020	
8.	Надання текстової частини керівнику для попередньої перевірки.	15.03.2020	
9.	Редагування текстової частини з урахуванням рекомендацій керівника.	23.03.2020	
10.	Створення слайдів для доповіді та написання доповіді.	квітень 2020	
11.	Захист курсової роботи.	квітень 2020	

Студент *Манжура А. В.*

Керівник *Жежерун О.П.*

“_____”_____

Зміст

Анотація

Метою курсової роботи є дослідження ефективних способів та засобів розробки баз знань у прикладних системах, таких як системи прийняття рішень. Об'єктом дослідження є поняття представлення знань, онтології, онтологічне моделювання, а також процес створення бази знань на основі цих засобів. Було розглянуто структуру онтологій, їх класифікацію, досліджено принципи побудови онтологічних моделей, детальний процес їх формування. Виконано огляд існуючих технологій для онтологічного моделювання, зроблено висновки щодо їх недоліків та переваг. У практичній частині було обрано найоптимальніше рішення для реалізації бази знань, розроблено базу знань медичного закладу на основі онтології в редакторі Protégé та протестовано її з використанням запитів SPARQL.

Вступ

Однією з найважливіших цінностей в сучасному інформаційному суспільстві є знання. Але чим знання відрізняються від звичного нам терміну дані? Різниця між двома цими словами полягає в наступному: для сприйняття та використання даних людині необхідно виконати інтерпретацію, тобто виокремити їх сенс і перенести на частину реальності, в якій вона зацікавлена; знання ж, що представлені в електронній формі, можуть сприйматись безпосередньо — без початкової обробки, бо вони вже і так виражені за допомогою понятійного апарату, яким людина послуговується. Тобто знання — це практично корисна інформація, яка вже сповнена сенсом і яку ми можемо використовувати в своїй діяльності. А для їх отримання необхідні ефективні технології обробки інформації та наповнення сирих даних певним сенсом.

На сьогодні управління знаннями є однією з ключових ідей, що впливають і на сучасні тенденції розвитку корпоративних систем і бізнесу. Діяльність як окремих людей, так і організацій зараз все більшою мірою залежить від наявних у них знань — одного з найцінніших ресурсів — і здатністю їх ефективно використовувати. Проте засоби, що призначені для представлення знань, ще недостатньо досконалі і часто змушують знов і знов шукати рішення одних і тих самих задач. Різноманітні групи користувачів, що займаються обробкою і аналізом інформації, використовують спеціальну термінологію, яка застосовується іншими спільнотами в іншому контексті. В той же час в різних спільнотах часто зустрічаються різні позначення для одних і тих самих понять. Все це суттєво ускладнює взаєморозуміння, тому важливо розробляти моделі представлення знань, які забезпечували б автоматизовану обробку інформації на семантичному рівні в так званих *системах управління знаннями* (СУЗ).

Управління знаннями — це сукупність технологій і процесів, які призначені для виявлення, створення, обробки, зберігання, поширення та надання знань для використання. На підприємствах ця практика застосовується для зберігання та обміну всією наявною у них інформації, досвіду та

кваліфікації співробітників для того, щоб підвищити рівень обслуговування клієнтів та скоротити час реакції на швидкоплинні ринкові умови. В таких випадках необхідно здійснити формалізований опис предметної області і сформувати її базу знань для подальшого використання цієї бази при вирішенні практичних задач, що виникають.

Одним з найважливіших напрямків в області формалізації знань, яке дає можливість використання накопичених знань для комп'ютерної обробки, є онтології. Онтологія – цілісна структурна специфікація певної предметної області, її формалізоване представлення, що описує поняття цієї області, а також відносини, що існують між цими поняттями. Таке представлення знань дозволяє і комп'ютеру, і людині ефективніше використовувати наявну інформацію.

Мета курсової роботи: розглянути переваги використання онтологій у проектуванні баз знань, дослідити процес і принципи побудови онтологічних моделей, виконати огляд існуючих технологій для онтологічного моделювання.

В результаті дослідження буде обрано найоптимальнішу технологію розробки з урахуванням різнобічних факторів, а також продемонстровано переваги та зручності використання онтологій в проектуванні баз знань на власноруч розробленій онтології для використання в медичних закладах.

РОЗДІЛ 1. ТЕОРЕТИЧНА ЧАСТИНА. ОНТОЛОГІЇ. БАЗИ ЗНАНЬ

1.1 Інтелектуальні корпоративні системи управління знаннями

Управління знаннями – це комплексна діяльність з організаційно-технічних аспектів, яка спрямована на те, щоб підвищити ефективність використання знань в бізнес-процесах корпорації. Цей термін почали використовувати ще з 1990-х років через проблеми, які виникли під час обробки великих обсягів інформації у великих компаніях. Він насамперед стосується процесів створення, розповсюдження, опрацювання та використання знань всередині підприємства.

Технічні характеристики сьогоднішньої апаратури надзвичайно потужні, проте навіть зараз для вирішення аналітичних задач, особливо у бізнес-середовищі, люди часто користуються в кращому випадку таблицями Excel або реляційними базами даних, а такі бізнес системи як ERP чи CRM не здатні надати організації-власнику нічого особливо інтелектуального. Це в першу чергу пов’язано з обмеженістю тієї частини автоматизованої обробки інформації, що освоїло бізнес-товариство. Навіть ті ефективні системи, що справді допомагають оптимізувати процеси в компанії, мають дуже вузьку галузеву спрямованість і містять алгоритми вирішення задач, що є жорстко запрограмованими, і таким чином комп’ютер насправді не в змозі по справжньому допомагати в вирішенні інтелектуальних бізнес-задач і не може підтримувати прийняття рішень в будь-яких сферах.

Постає питання: як навчити комп’ютер вирішувати задачі таким способом, що був би найбільш подібним до кращих способів, які використовують в аналітиці люди? Для цього необхідно відтворити в цифровому представленні такі інформаційні структури і процеси, якими ми самі користуємось під час мислення та прийняття рішень, а саме – понятійний апарат, логічне мислення, формування висновків. Тоді стане можливо реалізовувати процеси обробки

структур, тобто імітувати на комп’ютері окремі фрагменти наших когнітивних здібностей. Отримавши після цього певні результати і критично оцінивши їх, можливо буде самостійно поліпшити змодельовані структури і процеси. Разом з цим, враховуючи також недоступні людині машинні можливості до швидкої обробки великих обсягів інформації, такий підхід дозволив би мати дуже високий рівень якості підтримки прийняття рішень інформаційними системами.

Може здатись, що за допомогою одних лише логічних висновків з уже наявних фактів неможливо отримати якусь принципово нову інформацію. Проте впродовж усієї історії людства отримання висновків про нові феномени відбувалось через уподібнення їх вже відомим явищам. Отже, якщо доповнити імітацію строгого логічного мислення інструментами, які дозволяли б робити подібні висновки, можна було б отримати систему, що здатна до самонавчання та до вироблення принципово нових для нас знань.

1.1.1 Створення систем управління знаннями

1.1.1.1 Поняття знань, корпоративних знань, сховища даних

В СУЗ знаннями вважають усю доступну інформацію (документи, відомості про замовників, опис технологій роботи, продукції тощо), а також закономірності ПрО, отримані з практичного досвіду чи зовнішніх джерел. Одними із перших СУЗ були сховища даних. Потім ідея сховищ трансформувалась в поняття корпоративної пам’яті, яка містить інформацію з різних джерел і забезпечує доступ до неї спеціалістам для вирішення виробничих задач (Див. Рис. 1).

Рисунок 1 – Структура корпоративної пам'яті

Таблиця 1 ілюструє різницю між поняттями знання, корпоративні знання та сховище даних.

Таблиця 1 – Головні поняття в СУЗ

Поняття	Визначення
Знання	Сукупність відомостей, фактів, понять, представлень про що-небудь, які накопичуються внаслідок навчання, досвіду та загалом процесу будь-якої діяльності.
Корпоративні знання	Знання, які доступні організації в явному вигляді і можуть використовуватись співробітниками цієї організації для підвищення ефективності.
Сховище даних	Система зберігання даних великого обсягу, яка реалізована на основі баз даних різних типів і дозволяє об'єднувати їх в єдиний робочий масив.

1.1.1.2 Етапи розробки СУЗ

Під час розробки СУЗ виділяють кілька головних етапів, а саме таких:

а) накопичення – стихійне і безсистемне накопичення інформації в організації;

б) вилучення – процес, що є ідентичним до традиційного вилучення знань для експертної системи. Це є одним з найскладніших і найбільш трудомістких етапів. Від успішності цього процесу залежить і подальша життєздатність системи;

в) структурування – на цьому етапі мають бути виділені основні поняття, вироблена структура представлення інформації з максимальною наочністю, а також здатністю піддаватись простим змінам і доповненням;

г) обслуговування – коректування (оновлення чи додавання) формалізованих даних і знань, їх фільтрація для пошуку необхідної користувачу інформації, а також видалення застарілої інформації.

1.1.1.3 Джерела знань для СУЗ

При розгляді систем управління знаннями виникає цілком природне питання: звідки взагалі беруться знання? В ранніх експертних системах широко застосовувалось традиційне рішення – вилучення знань із пам’яті експерта. Для цього застосовувались і пасивні методи (спостереження, лекційні матеріали, аналіз протоколів роботи), і активні (інтерв’ю, анкетування, експертні ігри тощо). Але існує загальновідома закономірність: чим більше досвіду накопичує експерт, тим більше він втрачає здатність формалізувати ці знання в такому вигляді, що це було б доступне неспеціалісту. Саме тому працювати напряму з експертами настільки складно і дорого, адже для цього треба залучати кваліфікованих інженерів по знаннях. Ця задача також дуже трудомістка та потребує значної кількості часу. У зв’язку з цим для створення

прикладних систем, що основані на знаннях, доцільно використовувати методи автоматизованого видобутку знань з потоку даних, які показують реальну роботу спеціалістів в якій-небудь предметній області і дозволяють узагальнювати та формалізувати їхній досвід.

Data Mining (від англ. «видобуток даних») – напрямок в IT, який пов'язаний з інтелектуальним аналізом сховищ і баз даних, внаслідок чого витягуються знання, які неявно містяться в інформації, що обробляється, завдяки виявленим залежностям і тенденціям.

В основі сучасних технологій Data Mining лежить концепція шаблонів, які показують різноманітні фрагменти взаємозв'язків в даних. Важлива властивість методів Data Mining – нетривіальність знайдених шаблонів, які мають показувати неочевидні, раніше невідомі регулярності в даних, які складають так звані приховані знання (hidden knowledge). Прикладами сфер використання таких технологій є прогнозування змін клієнтури, аналіз кошика покупця в роздрібній торгівлі, виявлення шахрайства з кредитними картками.

Методи Data Mining дозволяють виділити такі типи закономірностей:

а) послідовність (наприклад після того, як клієнт купив товар А, з великою ймовірністю він купить і товар Б);

б) зв'язок між подіями (наприклад клієнт купує товари А і Б одночасно);

в) класифікація (клієнти даної компанії належать до однієї з груп, які мають відносно постійні потреби і вимоги);

г) кластеризація (відмінність від класифікації полягає в тому, що самі групи заздалегідь не задаються і виокремлюються безпосередньо в процесі аналізу);

ґ) прогноз – побудова часових рядів, що показують поведінку цільових показників.

Text Mining – один з підвидів Data Mining, яка орієнтована на обробку текстової інформації і представлення її в зручній користувачу формі. Широко застосовується для моніторингу ресурсів Інтернет.

Задачею Text Mining є проаналізувати не синтаксис, а семантику значень текстів, обрати з нього інформацію, що є найбільш цінною для користувача. Сфери застосування можуть бути такими:

а) подання текстів природною мовою;

б) тематичне індексування текстових документів;

в) кластеризація текстових документів і їх фрагментів;

г) побудова онтології текстового документа (основних термінів і зв’язків між ними), як-от семантичної мережі;

ґ) візуалізація отриманих знань.

1.1.1.4 Задачі і цілі онтологічного моделювання

Головна задача систем управління знаннями (СУЗ) – накопичувати не розрізнену інформацію, а структуровані, формалізовані знання – насамперед закономірності і принципи, які дозволяють вирішувати реальні робочі завдання. Основна ціль СУЗ – зробити так, щоб знання були доступними і придатними для повторного використання на рівні всієї компанії. А однією з найактуальніших проблем розвитку IT наразі є доведення до промислової експлуатації таких технологій, які б дозволили будувати складні комплексні моделі, за допомогою яких можна було б вирішувати оптимізаційні, аналітичні та оперативні задачі в таких СУЗ. Онтологічний підхід до проектування СУЗ якраз дозволяє створювати системи, в яких знання, що накопичені всередині організації, стають не лише доступними для більшості користувачів, а й можуть піддаватись виконанню абсолютно автоматичних операцій – отримання логічних висновків, в результаті яких будуть виведені нові знання.

Головною задачею онтологічного моделювання є створення формалізованих електронних моделей знань. Серед цілей їх використання варто виокремити такі:

а) швидке отримання логічних висновків на основі великої кількості інформації для того, щоб здійснити підтримку прийняття рішень;

б) виконання імітаційного моделювання процесів для того, щоб їх оптимізувати;

в) обмін знаннями між людьми за рахунок доступності для сприйняття великих обсягів структурованої інформації;

г) інтеграція інформаційних систем.

Наприклад, в електронній комерції онтологічні представлення знань використовують для підтримки автоматизованого обміну даними між покупцями і продавцями, для вертикальної інтеграції ринків, а також для повторного використання описів різними торгівельними точками. Механізми пошуку також використовують онтології для вибірки сторінок зі словами, що синтаксично різні, але семантично однакові. Важливість онтологічного підходу підкріплюється також і тим, що знання, яке не є описаним і тиражованим, зрештою стає застарілим та не корисним, а знання, яке розповсюджується, здобувається та яким обмінюються, генерує нове знання.

1.2.1 Принципи побудови онтологічних моделей

Під онтологічною моделлю розуміють концептуальне представлення інформації про якусь предметну область в електронному вигляді. Слово «концептуальне» підкреслює той факт, що модель будується за допомогою понятійного апарату. При моделюванні важливо чітко відокремлювати об’єкт (або явище) реального світу, його образ в нашій свідомості, поняття, яке використовується для позначення його образу, та ідею, що відповідає поняттю. Поняття використовуються для групування сутностей, які є однорідними в певному відношенні, як-от ті поняття, що володіють певним набором спільних ознак. Наприклад, верблюд для нас – це тварина з чотирма ногами, що має певну форму і пропорції тіла, морди, вух. Таким чином, можна чітко визначити, що концепт «верблюд» відповідає класу об’єктів реального світу. Належність конкретного об’єкту до класу робить можливим автоматичне надання об’єкту деяких властивостей, які можуть явно не міститись в отриманому повідомленні:

так, ми знаємо, що верблюд має горб, може довго жити без води тощо. Таким чином можна використовувати вже наявні знання для того, щоб записати сенс нового інформаційного повідомлення в загальну структуру наявних знань, завдяки чому повідомлення набуває нового змісту, формуючи нові знання.

Модель – інформаційне представлення якоїсь сукупності об’єктів і явищ реального світу, яке характеризується:

а) спрощеністю;

б) концептуалізованістю (кожен об’єкт виражений за допомогою понять різного рівня абстрактності);

в) взаємозв’язаністю;

г) наявністю логічних правил взаємодії елементів;

ґ) потенціалом прогнозування (за допомогою наявних в моделі параметрів та логічних правил можливо зробити висновок про те, що відбудеться в тому чи іншому випадку).

1.2.2 Процес формування моделі

Процес формування онтологічної моделі можна поділити на кілька етапів, а саме:

а) виділення об’єктів;

б) ідентифікація об’єктів;

в) класифікація;

г) опис властивостей.

1.2.2.1 Виділення об’єктів

Процес побудови інформаційної моделі будь-якої предметної області починається з декомпозиції, тобто розділення фрагменту реальності на окремі елементи – об’єкти, які стануть базовими одиницями моделі. Глибина декомпозиції залежить тільки від прагматики, тобто практичного призначення

моделі. Задачею правильного моделювання є пошук практично обґрунтованого і оптимального співвідношення між рівнем деталізації моделі і ресурсів, які для цього вимагаються. Кожен виокремлений об’єкт в процесі декомпозиції позначається терміном індивід.

Цілісна концептуальна модель має містити сутності таких типів: індивіди, класи об’єктів, визначення статичних атрибутів об’єктів та зв’язків між ними. Їх ще називають термінологією моделі, або TBox. В моделі, яка вирішує конкретні практичні задачі, будуть також міститись значення конкретних властивостей для конкретних індивідів, конкретні зв’язки між ними. Ця частина моделі називається ABox – набір тверджень або фактів. Третьою частиною повної онтологічної моделі може бут набір правил отримання логічних висновків.

1.2.2.2 Ідентифікація об’єктів

Наступним етапом є ідентифікація об’єктів, тобто їх однозначне визначення шляхом надання об’єкту унікального ідентифікатора. Важливо при цьому дотримуватись принципу нейтральності ідентифікатора, а саме щоб він не містив в собі якогось смислу і не базувався на жодних властивостях об’єкту.

1.2.2.3 Класифікація

Третім кроком є класифікація об’єктів, тобто віднесення кожного об’єкту до одного або декількох класів. Клас слугує для загального позначення якоїсь сукупності об’єктів. За допомогою тверджень, як-от «Усі члени класу X є членами класу У» – можливо будувати ієрархію класів, або таксономію. Кожен індивідуальний об’єкт також можна віднести одночасно до декількох класів; такий принцип класифікації відомий як фасетний. Розділення на класи та індивіди, як і визначення ієрархії, залежить від контексту використання моделі, тож не існує однозначно правильної чи неправильної класифікації.

1.2.2.3 Опис властивостей

Модель, яка складається лише з класів та індивідів, має мало практичного змісту, тому треба приписати об’єктам ті чи інші властивості, адже правила взаємодії об’єктів майже завжди залежать від значень цих властивостей. Основні характеристики властивості включають:

а) назву;

б) обмеження на тип і діапазон значень;

в) набір об’єктів, які можуть або повинні мати цю властивість.

1.2 Онтології. Технології для онтологічного моделювання

Термін «онтологія» походить з гілки філософії, відомої як метафізика, яка є вченням про природу реальності і буття. Вона займається аналізом різних типів та форм існування з урахуванням їх особливостей та відносин між сутністю та самим існуванням; має за мету класифікацію світу для виокремлення фундаментальних категорій або видів, до яких належать об’єкти у світі.

Онтологія в інформатиці – це детальний опис деякої області знань, який використовується для формального визначення її концептуалізації. Під формальною концептуалізацією розуміють певну абстракцію або спрощений вигляд предметної області, який є зрозумілим та читабельним для обчислювальних машин і який містить строгі визначення типів, властивостей і взаємозв’язків об’єктів в ПрО, тобто базис для моделювання даної області знань. Іншими словами онтологія – точна специфікація певної ПрО, яка містить словник термінів і набір логічних зв’язків для опису взаємодії цих термінів між собою.

Таким чином онтологія в інформатиці є практичним застосуванням філософського поняття онтології за допомогою таксономії, тобто класифікації та систематизації складноорганізованих сутностей, що ієрархічно співвідносяться. Спільнота розробників почала використовувати термін онтології з кінця 20

століття для того, щоб позначити теорію модельованості світу і компонента системи знань.

В інформаційних галузях онтології створюються для цілісного представлення деякої предметної області, яке було б придатне для комп'ютерної обробки. Вони використовуються як посередник між інформаційною системою та кінцевим користувачем, адже за допомогою неї можна формалізувати термінологію і певні домовленості про теоретичні основи між членами спільноти, наприклад між користувачами корпоративного сховища даних. На Рисунку 2 зображена схема використання онтології в інформаційних системах.

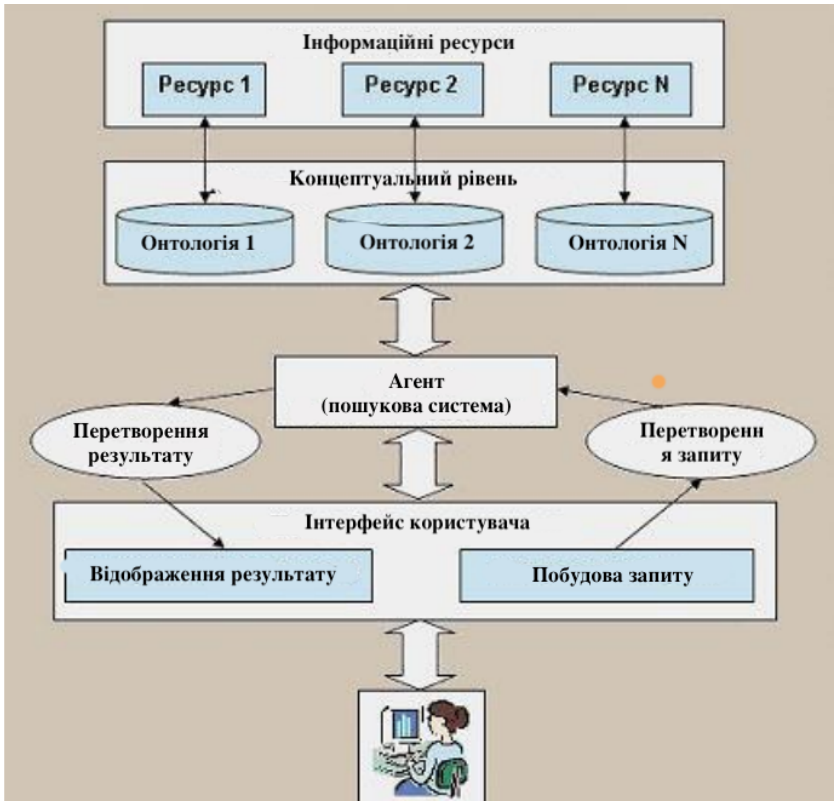


Рисунок 2 – Схема використання онтології в інформаційних системах

Серед основних переваг варто виокремити такі:

а) онтологія надає користувачу цілісний, системний погляд на певну предметну область (ПрО);

б) знання про ПрО представлені однорідно, що спрощує їх сприйняття;

в) побудова онтології дозволяє відновити логічні зв'язки ПрО, яких не вистачає.

1.2.1 Основні компоненти онтологій

Онтології містять такі загальні компоненти:

а) класи (концепти) – представлення типів об’єктів в предметній області. Об’єкти можуть бути і конкретними речами, і абстрактними поняттями;

б) індивіди (екземпляри) – представлення конкретизації класу, тобто конкретний об’єкт;

в) відносини – представлення взаємозв’язків між концептами (чи окремими об’єктами) в ПрО. Розрізняють 2 основні типи відносин:

1) таксономічні – відносини, що організовують концепти в ієрархії (наприклад такі стосунки, як «є підкласом», «є підтипом», «є частиною»);

2) асоціативні – співвідносять поняття в різних ієрархіях концептів (наприклад «є наслідком», «спричинений»).

г) атрибути (параметри, слоти) – властивості або параметри об’єктів, які клас може мати;

ґ) функціональні терміни – побудовані з певних відносин складні структури, якими можна замінити окремий термін;

д) аксіоми – твердження, які завжди правдиві в предметній області. Використовуються для визначення обмежень та правил у логічній формі. Ці види тверджень є корисними для переконання у послідовності знання в онтології, а також для отримання нових знань, які до цього не були явно визначені в онтології.

1.2.2 Використання онтологій

Онтології мають багато задач та цілей, як-от:

а) аналіз знань в ПрО і на основі отриманої логічної теорії виведення нових знань, що початково не були закладені в онтологію;

б) надання можливості для повторного використання та обміну знаннями;

в) створення основи взаємодії між системами;

г) розробляти термінологію і управляти нею;

ґ) відокремлювати знання в ПрО від оперативних знань;

д) спільне використання користувачами загального розуміння структури інформації.

Останній пункт є зазвичай головною метою розробки будь-яких онтологій. Адже якщо всі користувачі спільно користуються однією і тією самою базовою онтологією термінів, то стає можливим не тільки накопичення інформації комп'ютерними агентами, але й використання їх для формування відповідей на запити користувачів або як вхідні дані для інших систем.

Серед сценаріїв використання онтологій варто також виділити такі:

а) як спосіб комунікації між людьми та організаціями;

б) як спосіб інтеграції даних;

в) як сховище інформації та спосіб оранізації знань;

г) як засіб підтримки функціонування нових видів цифрових бібліотек, які реалізовані у вигляді розподілених інтелектуальних систем.

Існує безліч сфер використання, де онтології слугують ключовою технологією, наприклад:

а) семантичний веб;

б) програмна інженерія – як засіб представлення різних артефактів процесу розробки на будь-якому етапі життєвого циклу;

в) штучний інтелект – як засіб представлення загальних знань;

г) комп'ютерна безпека – для кодування властивостей ресурсів та різноманітних загроз;

ґ) біомедицина – як сховища знань та як засіб інтеграції даних в неоднорідних джерелах;

д) імітаційне моделювання;

ж) експертні системи – підтримка прийняття рішень.

1.2.3 Структура онтологій

У загальному випадку структура онтології складається з набору елементів чотирьох категорій: класи, відносини, аксіоми і окремі екземпляри.
Класи – основні компоненти, на яких будується онтологія. Вони є загальними категоріями, що можуть бути ієрархічно впорядкованими. Класи описують певну групу конкретних сутностей зі спільними властивостями. Наприклад, клас Книги репрезентує усі книги. Конкретні книги є екземплярами цього класу. Книга «Основи програмування», що лежить на столі, є екземпляром (представником) класу Підручників з програмування.

Класи (або поняття) в онтологіях пов’язані між собою різними відносинами. Найпоширенішим видом відносин є категоризація, ще відома як таксономічне відношення, тобто відношення типу клас-підклас або ставлення IS-A. При таких відносинах клас має підклас, який репрезентує поняття з більшою конкретикою ніж батьківський клас. Наприклад, ми можемо поділити клас всіх книжок на художню літературу та на нехудожню. Або, як альтернатива, – на книги для дорослих і на книги для дітей.

Аксіоми то є вираженням очевидних тверджень, що зв’язують класи і відносини. З однієї аксіоми в онтології можна логічно вивести інші твердження. За допомогою них можна надати ту інформацію, яку неможливо висловити лише ієрархією класів та відносинами між ними. Прикладом аксіоми слугує таке твердження: «Якщо Y смертний, то це значить, що Y колись помре». Аксіоми можуть додатково надавати інформацію про певні правила чи обмеження, роблячи можливим виведення висновків на їх основі. Прикладами обмежень є поняттєві (властивість Матеріал може виражатись лише поняттям категорії Матеріал) та числові (твердження, що у Людини кількість біологічних батьків строго рівна двом).

Рисунок 3 ілюструє приклад структури онтології.

Рисунок 3 – Приклад структури онтології

Самі властивості також можуть бути пов’язаними між собою якимись відносинами, а також володіти характеристиками, які дозволяють роботи логічні висновки. Варто перерахувати основні з таких властивостей:

а) еквівалентність;

б) інверсивність;

в) роз’єднаність (наявність однієї властивості виключає наявність другої);

г) транзитивність (якщо А пов’язаний з Б, а Б – з В, то А пов’язаний з В);

д) симетричність (якщо А пов’язаний з Б, то і Б пов’язаний з А).

1.2.4 Класифікація онтологій

Існує безліч різних класифікацій онтологій. Деякі розрізняють їх за рівнем залежності від ПрО чи конкретної задачі, за набором елементів в онтології та відносин між ними, за мовою представлення знань, за рівнем деталізації аксіом, за формою, за змістом, за засобами використання онтологій тощо. Наявні й інші поділи на різні типи, наприклад згідно з Е. Хові можна виокремити «справжні» та «термінологічні» онтології. Вони різняться кількістю компонентів, що входять до них, та різними типами відносин. Але загальноприйнятою основною класифікацією є наступна:

а) верхні (загальні) онтології;

б) предметно-орієнтовані онтології;

в) онтології, орієнтовані на прикладну задачу;

г) прикладні онтології (як об’єднання попередніх двох).

Рисунок 4 показує як ці типи онтологій співвідносяться один з одним з точки зору зручності використання та можливості повторного використання.

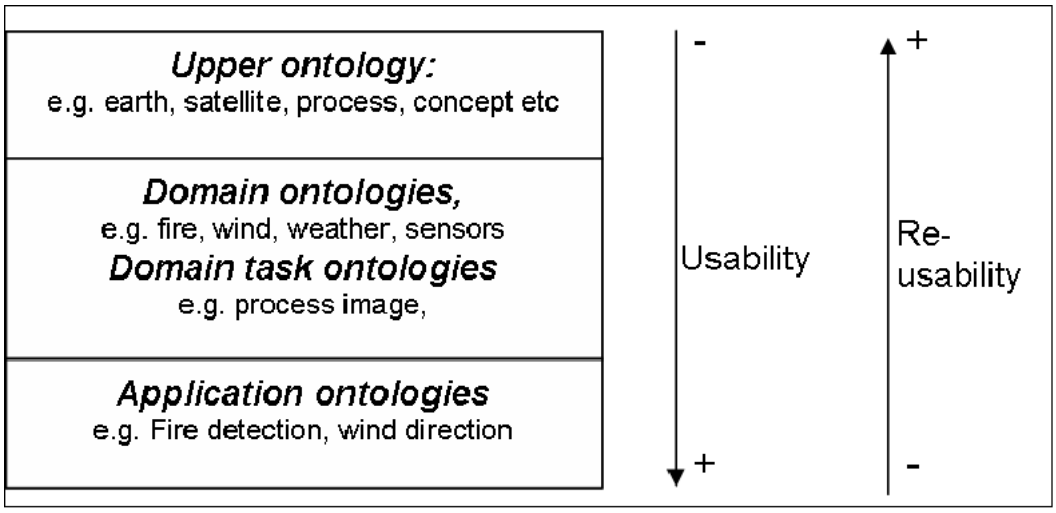


Рисунок 4 – Типи онтологій

1.2.4.1 Предметно-орієнтовані онтології

Такі онтології використовуються експертами в своїх предметних областях за рахунок того, що там містяться класи, які належать до певної області і, відповідно, там моделюються терміни зі значенням, що стосується конкретно цієї області. Наприклад, термін «карта» є багатозначним, його визначення залежить від онтології, в якій він використовується: в онтології «Хвороби» термін «карта» буде моделювати різні симптоми, а в онтології «Покер» – різні значення гральних карт («дама піка», «король бубна»).

1.2.4.2 Онтології, орієнтовані на прикладну задачу

Головна мета таких онтологій є уможливлення повторного використання знань в ній. Вони використовуються конкретною прикладною програмою і тому відображає її специфіку через терміни, які виконують конкретну задачу. Наприклад, в онтології «Обробка зображень» будуть присутні специфічні терміни «заливка», «растрування», «накладання шарів». Визначається поняття дії та забезпечується її моделювання. Онтології цього типу широко застосовуються для проектування інтерфейсів комп'ютерних програм, в експертних системах, системах підтримки прийняття рішень.

1.2.4.3 Верхні онтології

Верхні онтології описують найбільш загальні класи, незалежні від якоїсь предметної області чи задачі. Прикладами таких онтологій слугують WordNet та CYC. В таких онтологіях зазвичай містяться найбільш абстрактні категорії та обмежений набір базових відносин з кількістю концептів в діапазоні від 100 до 500, а також обов'язково ядро-словник з термінами і описами об'єктів для використання в найрізноманітніших предметних областях. Така онтологія вміщує описані вище два типи онтологій – предметно-орієнтовані та орієнтовані на прикладну задачу.

1.2.5 Мови опису онтологій

Формальна мова для кодування онтології називається мовою опису онтологій. Такі мови дозволяють реалізовувати різні онтології без необхідності розв'язування низькорівневих проблем в процесі. Існує кілька подібних мов:

а) OWL (Ontology Web Language) – стандарт W3C, створена для семантичних тверджень, є розширенням RDF і RDFS. Рисунок 5 ілюструє приклад коду, написаного цією мовою;

```
    <owl:Class>
        <owl:unionOf rdf:parseType="Collection">
             <owl:Class rdf:about="#MozzarellaTopping"/>
             <owl:Class
            rdf:about="#PeperoniSausageTopping"/>
             <owl:Class rdf:about="#JalapenoPepperTopping"/>
             <owl:Class rdf:about="#TomatoTopping"/>
             <owl:Class rdf:about="#HotGreenPepperTopping"/>
        </owl:unionOf>
    </owl:Class>
```

Рисунок 5 – Приклад мови OWL (онтологія Піца)

б) KIF (Knowledge Interchange Format) – заснований на спеціалізованому синтаксису для логіки. Рисунок 5 показує приклад аксіоми онтології, вираженої на KIF;

```
  (subclass RailVehicle LandVehicle)
     (documentation RailVehicle
          "A Vehicle designed to move on &%Railways.")
     (=>  (instance ?X RailVehicle)
          (hasPurpose ?X
           (exists (?EV ?SURF)
             (and  (instance ?RAIL Railway)
                   (instance ?EV Transportation)
                   (holdsDuring (WhenFn ?EV)
                                (meetsSpatially ?X ?RAIL))))))
```

Рисунок 6 – Приклад мови KIF

в) CycL – заснована на численні предикатів з деякими розширеннями більш високого порядку;

г) DAML+OIL(FIPA);

ґ) DOGMA (Developing Ontology–Grounded Methods and Applications);

Д) RDF (Resource Description Framework) має мету додати формальну семантику вебу та надати модель даних у стандартизованому вигляді. Рисунок 7 ілюструє приклад опису магазину платівок на мові RDF. RDF Schema (RDFS) – спеціальний стандарт для представлення онтологічних знань, але без можливості виразити аксіоматичні знання.

```
<rdf:Description
rdf:about="http://www.recshop.fake/cd/Hide your heart">
    <cd:artist>Bonnie Tyler</cd:artist>
    <cd:country>UK</cd:country>
    <cd:company>CBS Records</cd:company>
    <cd:price>9.90</cd:price>
    <cd:year>1988</cd:year>
</rdf:Description>
```

Рисунок 7 – Приклад коду на RDF

Таблиця 2 показує порівняння мов KIF, OWL, RDF+RDF(S), DAML+OIL за вказаними критеріями.

Таблиця 2 – Порівняння описових мов онтологій

Критерій	**KIF**	**OWL**	**RDF+RDF(S)**	**DAML+OIL**
Структура чи концептуалізація ПрО	Трьохрівневий синтаксис: базові символи; лексеми з цих символів; та вирази, складені з лексем.	Основі компоненти: клас, що визначає групу індивідів зі спільними властивостями; індивіди – екземпляри класів.	Триплет виразів: об’єкт, предикат (властивість), значення.	Структура описана класами і властивостями
Виразність	Висока	Залежить від підмови: Lite(низька) , DL(середня), Full(висока).	Середня	Види аксіом та види конструктора класів.
Перевірка обмежень	Слабка	Хороша	Слабка	Середня

Приведення до інших мов	Спроектована для використання у взаємодії розрізненних комп'ютерних системах.	RDF	OWL, DAML+OIL	RDF
Парадигма	Логіка предикатів першого рівня	DL	Об'єктно орієнтована структура	DL
Веб-стандарт	Нема	RDF	XML	XML та RDF
Популярність	Низька	Дуже висока	Середня	Низька

Як видно з Таблиці 2 найоптимальнішою мовою для представлення онтології у вебі наразі є мова OWL, яка дає можливості потужного семантичного аналізу. Вона є розширенням усіх попередніх мов, усуваючи більшість їх обмежень. Онтології OWL є по своїй суті документами WEB, на них можна посилатись через URI, вони містять послідовність аксіом і фактів, а також посилання на інші онтології. Таким чином, такі онтології є ключовою технологією для використання у семантичному вебі, а саме в організації обробки знань та для їх спільного використання.

Рисунок 8 показує ієрархію мов онтологій для семантичного вебу.

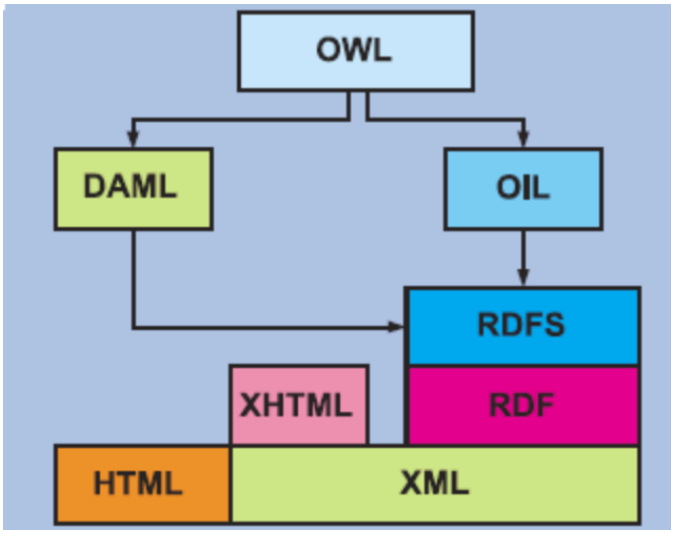


Рисунок 8 – Мови онтологій для семантичного вебу

1.2.6 Редактори онтологій

Існує цілий ряд інструментального програмного забезпечення, за допомогою якого можна редагувати, візуалізувати, документувати, імпортувати та експортувати онтології в різних форматах, поєднувати їх та порівнювати. Серед них варто виділити такі:

а) Apollo. Дозволяє користувачеві моделювати онтологію з базовим примітивами, такими як класи, екземпляри, функції, відносини тощо. Внутрішня модель – це фрейм система, заснована на протоколі OKBC. База знань в такому редакторі складається з ієрархічної організації онтологій. Онтології можуть бути успадковані від інших онтологій. Кожен клас може створювати ряд екземплярів, а екземпляр успадковує всі властивості класу. Проте Apollo не підтримує представлення у вигляді графу, веб, витяг інформації та можливості спільно використовуватись багатьма користувачами одночасно, але вона містить сильну перевірку відповідності типів, зберігання онтологій у файлах та формати для імпорту/експорту. Рисунок 9 демонструє інтерфейс Apollo.

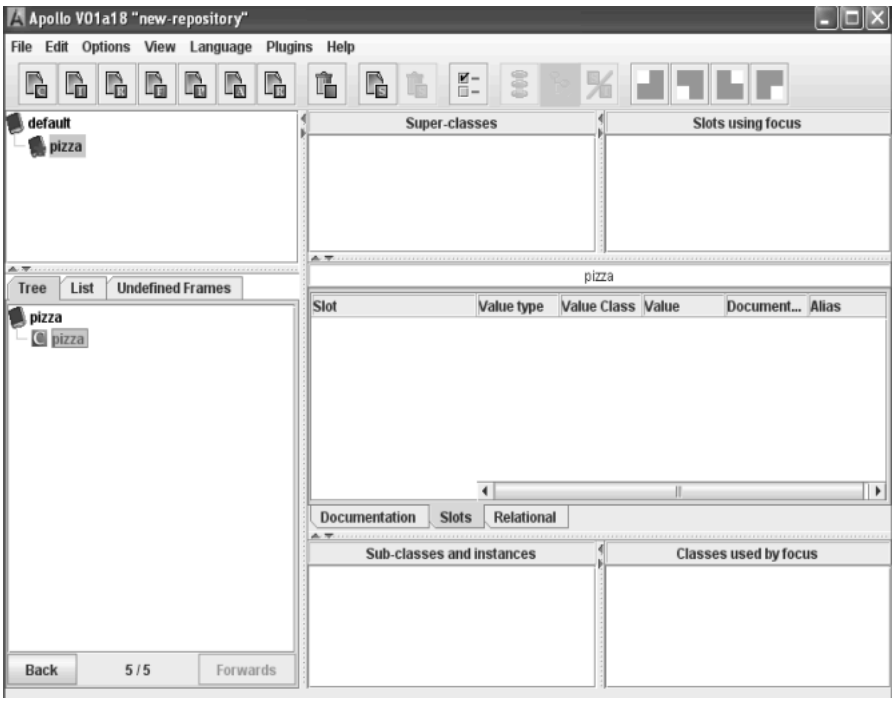


Рисунок 9 – Скріншот з Apollo

б) OntoStudio. Заснований на IBM Eclipse фреймворку. Це середовище інжинірингу онтологій, що дозволяє розробляти та підтримувати онтології з використанням графічних засобів. Воно базується на клієнт-серверній архітектурі, де онтології управляються в центральному сервері і різні клієнти можуть отримати доступ і вносити зміни в ці онтології. OntoStudio підтримує багатомовну і спільну розробку. Цей редактор дозволяє користувачеві редагувати ієрархію понять або класів. Внутрішні представлення моделей даних можна експортувати в DAML + OIL, F-Logic, RDF (S), OXML, а також до реляційних баз даних через JDBC. OntoStudio може імпортувати зовнішні представлення даних в форматах DAML + OIL, Excel, RDF (S), схемах баз даних (Oracle, MS-SQL, DB2, MySQL), OXML, а також файлів OWL. Більше того, OntoStudio надає API для доступу до онтологій у об’єктно-орієнтований спосіб. Рисунок 10 показує зовнішній вигляд інтерфейсу OntoStudio.

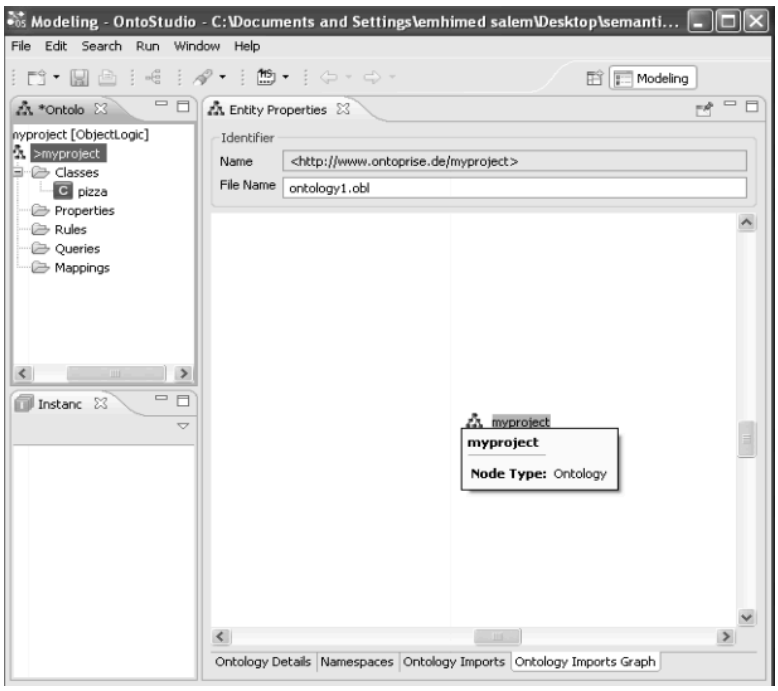


Рисунок 10 – Скріншот з редактора OntoStudio

в) Protégé – це вільна платформа з відкритим кодом, яка підтримується великою спільнота користувачів та надає набір інструментів для побудови модель предметної області та бази знань з онтологіями. Вона пропонує багато структур для моделювання знань та дозволяє створювати, візуалізувати та маніпулювати онтологіями в різних форматах представлення. Protégé може бути розширеною через архітектуру плагінів та API, що основане на Java. Protégé дозволяє визначати класи, змінні ієрархії класів, обмеження значень змінних і відносини між класами, а також властивості цих відносин. Рисунок 11 ілюструє інтерфейс Protégé.

ProtégéOWL надає API, яке може доступатись до зовнішнього DIG-сумісного семантичного механізму формування міркувань, що дозволяє робити висновки про класи та індивіди з онтології. Protégé також включає інтерфейс для SWRL (Semantic Web Rule Language), яка базується на OWL для виконання математики, тимчасових міркувань та додає правила міркування типу Prolog.

Істотною перевагою Protégé є його масштабованість та розширюваність. Protégé дозволяє ефективно будувати та обробляти великі онтології. Через свою

розширюваність Protégé може бути налаштовано відповідно до потреб та вимог користувачів.

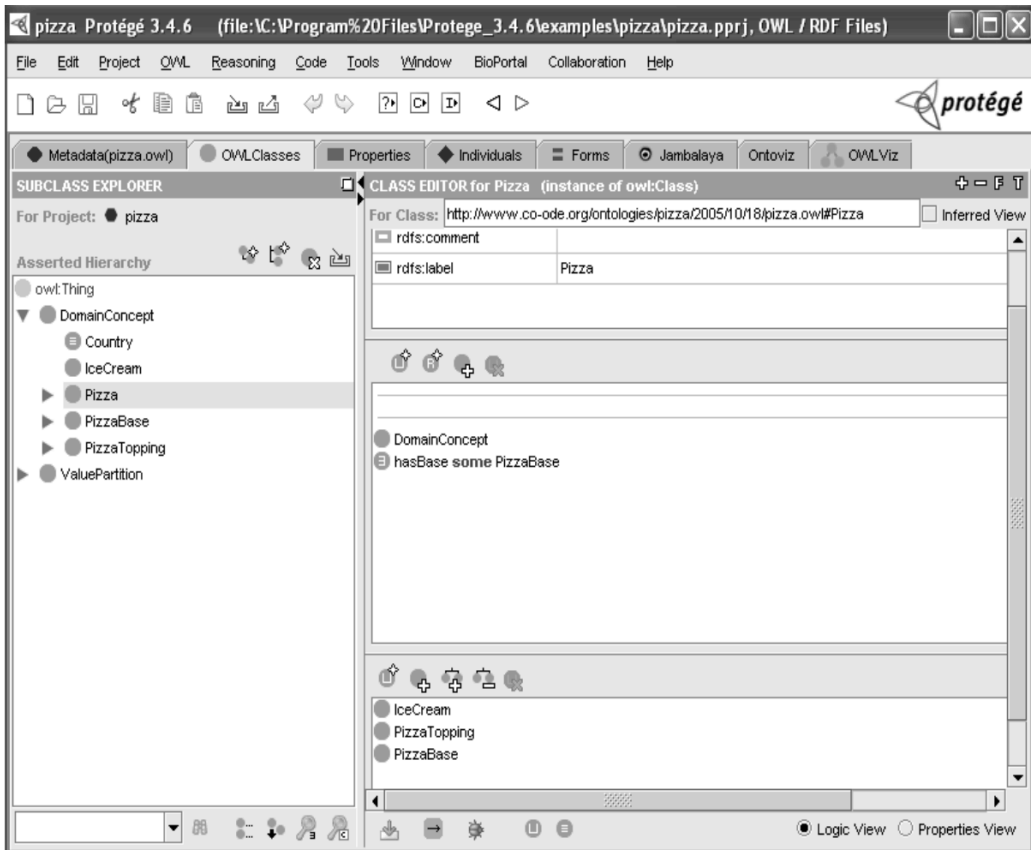


Рисунок 11 – Скріншот з редактора Protege

Таблиця 3 – Порівняння редакторів онтологій

Критерій	**Apollo**	**OntoEdit**	**Protégé**
Доступність	Open source	Необхідна ліцензія	Open Source
Семантична веб архітектура	Самостійна	Клієнт-серверна на Eclipse	Самостійна/ Клієнт-серверна
Зберігання онтологій	Файли	DBMS	Файли/ DBMS(JDBC)
Імпорт з	Apollo мета мови	XML(S), OWL, Excel, RDF(S), Outlook emails	XML(S), OWL, Excel, RDF(S), HTML, текстовий

			файл, Excel, BioPortal та DataMaster
Експорт в	OCML, CLOS	OWL, RDF(S), RIF, SPARQL, F-Логіка та Excel	XML(S), RDF(S), OWL, HTML, Java, F-Логіка, OWLDoc, Queries
Графічна таксономія	Ні	Так	Так
Спільна розробка	Ні	Так	Так

Таблиця 3 містить порівняння цих трьох редакторів. З неї можна зробити висновок, що редактор онтологій Protégé є найбільш оптимальним рішенням для розробки онтологій з огляду на дуже зручний графічний дизайн, кастомізацію, розширюваність, можливість спільної розробки, відкритого коду, а також великого різноманіття форматів та мов, з яких можна імпортувати та в які можна експортувати побудовані в Protégé онтології.

1.2.7 Різонери

Машини логічного виведення, або різонери, – клас програмного забезпечення, який призначений для роботи з онтологічними моделями. Вони дозволяють вираховувати значення логічних виразів, перевіряти правильність онтології і автоматично насичувати її новою інформацією відповідно до правил. Серед найпопулярніших різонерів можна згадати такі:

а) Pellet;

б) Racer;

в) FACT++;

г) Hermit.

РОЗДІЛ 2. ПРАКТИЧНА ЧАСТИНА. РОЗРОБКА КОРПОРАТИВНОЇ БАЗИ ЗНАНЬ

2.1 Визначення предметної області та масштабу бази знань

Перед початком розробки бази знань, необхідно визначитись з тим, в якій сфері її доцільно використовувати та яку користь вона може принести. Тобто треба дати відповіді на такі питання:

а) Яку предметну область база знань буде охоплювати?

б) З якою метою і ким база знань буде використовуватись?

в) На які питання база знань зможе давати відповіді користувачам?

Предметна область моєї системи – медичний заклад. База знань буде використовуватись для пошуку пацієнтів, лікарів, палат, симптомів. Проте найголовнішою перевагою моєї бази знань і тим, що відрізнятиме її від звичайної бази даних, буде функціонування її також як і системи підтримки рішень, експертної системи. База знань буде використовуватись працівниками медреєстратури, лікарями, адміністраторами та керівниками медзакладу.

Масштаб бази знань визначається списком питань, на які вона має надавати відповіді. Серед можливостей пошуку буде реалізовано надання відповідних знань як відповідей на такі питання:

а) Скільки у лікарні лежить людей з проблемами з диханням?

б) Скільки разів апарати штучної вентиляції легень використовувались пацієнтами і чи не варто його вже замінити, щоб закупити новий?

в) По яких робочих днях працює ваш сімейний лікар і ким його можна замінити на випадок відсутності?

г) На яке вільне ліжко можна покласти нового пацієнта та у яку палату?

ґ) Яка кількість договорів укладена у даного сімейного лікаря?

д) Яка кількість зайнятих ліжок у палаті/лікарні/районі/місті?

ж) На яких ліжках вже треба замінити білизну?

з) Які палати треба прибрати прибиральниці?

2.2 Технології розробки

Я буду проектувати свою базу знань за допомогою семантичних технологій. Ці технології є набором способів представлення і використання концептуалізованої інформації в електронному вигляді. Перевагою над простими реляційними базами даних є їхня можливість втілити в електронному вигляді концептуальні моделі, побудовані за принципами, описаними в попередньому розділі, адже вони дозволяють передавати і автоматично обробляти інформацію, що міститься в цих моделях, в тому числі і отримувати певні логічні висновки на основі правил.

Для реалізації моєї бази знань буде розроблено онтологічну модель в редакторі Protégé, використовуючи мови запису RDF, RDFS та OWL. Також мною буде використано різонер Pellet в якості машини логічного виведення, щоб формувати висновки на основі вже наявних тверджень та правил SWRL, а також перевіряти онтологію на несуперечливість та узгодженість. Для добування знань з бази будуть використовуватись точки доступу SPARQL, що дозволяють виконувати запити до онтологічних моделей. Для того, щоб отримані запитами знання включали також нову інформацію, сформовану на основі правил та онтологічних залежностей та відносин між об’єктами, в редакторі Protégé буде підключено плагін Snap SPARQL Query. Також буде використано плагін OntoGraf для інтерактивної навігації у відносинах в онтології та візуалізації цих відносин у вигляді графу.

2.3 Опис реалізації

Основними етапами створення бази знань будуть такі:

1) Визначення класів онтології;
2) Створення ієрархії класів;
3) Визначення властивостей, відносин, їх типів та обмежень;
4) Заповнення бази екземплярами та встановлення зв’язків між ними.

2.3.1 Визначення класів та ієрархії класів онтології

В своїй розробці я скористалась комбінованим підходом до створення онтології. Комбінований підхід полягає в поєднанні висхідного і низхідного способу проектування. Висхідна розробка передбачає початкове визначення найконкретніших класів, а потім вже групування їх в більш загальні поняття. Наприклад, я почала зі створення класів Клініка, Лікарня, Місто, Країна, Район, Палата. Потім створила надклас для перших двох термінів – Організація. Потім об'єднала ці класи ще одним загальним надкласом – Місце. Також я створила конкретні класи Доктор, Пацієнт, Робітник, об'єднавши їх спільним надкласом Людина. Так само, маючи конкретні терміни Ліжко, Крапельниця, Глюкометр, Апарат штучної вентиляції легень, я надала їм всім поняття вищого рівня – Обладнання. Після цього я скористалась низхідним підходом, а саме ще більша конкретизація загальних понять, тобто створення підкласів. Я взяла загальний клас Доктор та створила більш конкретні приклади цього поняття, тобто нові класи: Сімейний лікар, Головний лікар, Кардіолог, Пульмонолог. Також було конкретизовано поняття Пацієнт і створено підкласи: Пацієнт з жаром, Пацієнт зі складністю дихання. Так само з'явились підкласи у Ліжка – Зайняте ліжко і Вільне ліжко, а також у Палати – Пуста палата і Непуста палата.

Варто зазначити, що певні логічні висновки онтологія вже здатна робити лише спираючись на певні особливості ієрархічних відносин. Наприклад, такі відносини є транзитивними, тобто, якщо Сімейний лікар є підкласом Лікаря, а Лікар є підкласом Людини, то всі екземпляри класу Сімейний лікар вже автоматично вважаються екземплярами класу Людина і наслідує автоматично усі його властивості, проте також може володіти додатковими властивостями або обмеженнями, які не має суперклас.

Таким чином мною було сформовано ієрархію класів, як зображено на Рисунку 12.

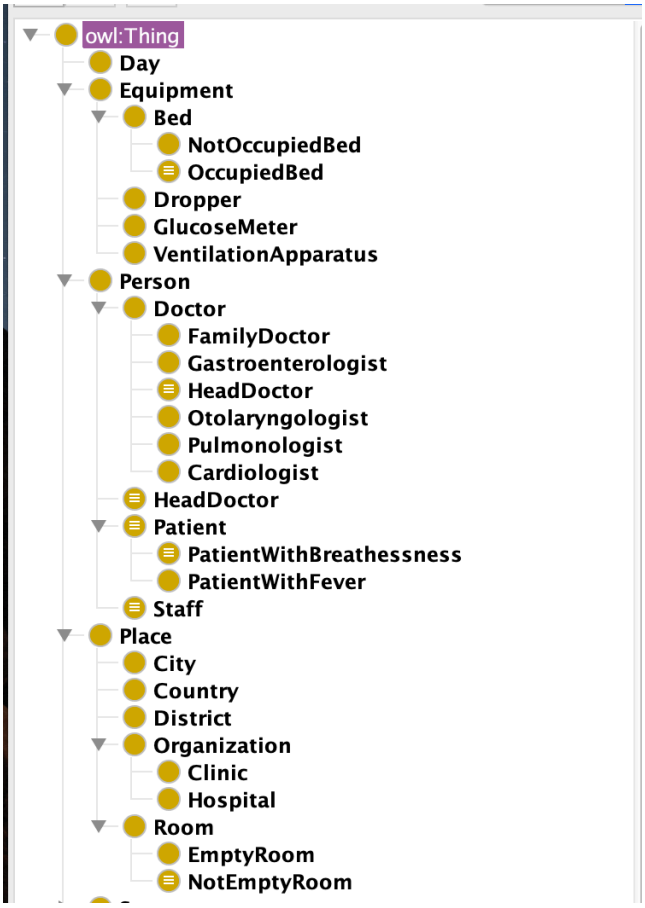


Рисунок 12 – Скріншот з Protege з ієрархією класів

2.3.2 Визначення властивостей

Усі властивості при розробці класів можна поділити на такі, що мають значеннями якісь певні типи даних (рядок, цифра, булеан), та на такі, що мають значенням інші класи. При цьому кожній властивості вказується її домен, тобто класи, які можуть цією властивістю володіти. Наприклад, властивість «ім’я» має тип значення string і домен Людина, тобто фактично це означає, що будь-який екземпляр класу Людина і її підкласів можуть володіти якимсь ім’ям. Так само із прізвищем та віком. Окрім цього я також створила властивості щодо симптомів захворювань, а саме «кашляє», «має проблеми з диханням, які можуть мати значеннями boolean, та «має температуру», що маже мати значенням double. Більше того, я виокремила властивості, що будуть

автоматично надаватись об’єктам в залежності від певних правил чи логічних висновків. Їх можна назвати властивостями, які уособлюють собою рекомендації для кінцевого користувача. Наприклад, клас Ліжко (а також його підкласи) може мати властивість «має бути замінено спальною білизною», а Кімнати може мати властивість «має бути прибраною». Всі перераховані вище приклади належать до типу Data properties. Повний їхній перелік можна побачити в додатку А.

Object properties, тобто ті властивості, що мають значеннями інші класи, можуть володіти ще окремо різними характеристиками, що дозволяють робити логічні виведення нових відносин (див. повний перелік в додатку Б). За цими характеристиками відносини між класами можуть бути:

а) функційними (один об’єкт може мати лише один екземпляр такого зв’язку з іншим окремим екземпляром). Прикладом в моїй онтологій може бути зв’язок «має палату» (виділена властивість на Додатку Б). І це логічно, адже один пацієнт може займати лише одну палату в лікарні. Різонер не дозволить нам створити ще один такий зв’язок в даного пацієнта.

б) інверсійно функційним (обернена функційна властивість до іншої). Наприклад, екземпляр класу Обладнання може мати зв’язок «використовується для», який є оберненим до зв’язку «використовує». Це значить, що якщо певний пацієнт «використовує» якесь певне обладнання, наприклад апарат штучної вентиляції легень, то у самого цього апарата автоматично формується інверсійно функційний зв’язок «використовується для» цього конкретного пацієнта.

в) транзитивними. Наприклад, якщо пацієнт «знаходиться в» лікарні Святого Павла, а лікарня Святого Павла «знаходиться в» Шевченківському районі, а Шевченківський район «знаходиться в» Києві, то тоді пацієнт «знаходиться в» Шевченківському районі і в Києві.

г) симетричними. Наприклад, якщо сімейний лікар Марія Василівна має властивість «може бути замінений на» сімейного лікаря Анна Федорівну, то

значить і Анна Федорівна має зв’язок «може бути замінений на» лікаря Марію Василівну у разі її відсутності на робочому місці.

ґ) асиметричні. Якщо пацієнт Федір «має контракт з сімейним лікарем» Анною Федорівною, то різонер не дасть створити Анні Федорівні той самий зв’язок. В моїй онтологій тоді в Анни Федорівни створиться автоматично інверсійний зв’язок «має контракт з пацієнтом» Федором.

д) рефлексивні. Якщо об’єкт А має зв’язок Е з об’єктом С, значить об’єкт А має зв’язок Е сам із собою.

ж) нерефлексивні. Якщо доктор Василь «працює в» лікарні Святого Павла, то різонер не дасть створити лікарні Святого Павла зв’язок «працює в» докторі Василі.

Також, окрім різних характеристик властивостей, ці властивості можуть також формувати ієрархії, подібно до класів. Так якщо «працює в » є батьківським зв’язком до «є головним лікарем в », то вказавши доктору Анні Василівні властивість, що вона «є головним лікарем в» лікарні Святого Павла, різонер автоматично сформує для неї нову властивість – що вона «працює в» лікарні Святого Павла.

2.3.3 Визначення обмежень та еквівалентів класів

Створюючи певний екземпляр нам не обов’язково вказувати напряму, членом якого класу він є. Система сама здатна визначити до якого класу віднести об’єкт та якими подальшим властивостями наділити. Для цього в онтології треба початково задати еквіваленти клсів, тобто певні твердження, які в разі правильності для даного об’єкта дозволяють автоматично віднести його до цього класу. Наприклад, якщо якась людина має зв’язок «є головою» якоїсь клініки чи лікарні, то ця людина автоматично визначається онтологією як Головний лікар (див. Рисунок 12). Або якщо якась Палата «має лежачого в ній пацієнта», то система автоматично визначає цю палату, як Непусту палату, а

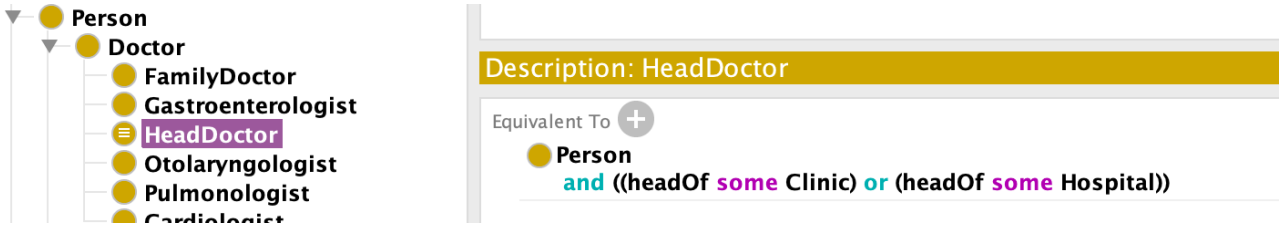


Рисунок 13 – еквівалент класу Головний лікар

оскільки вказано, що всі непусті палати мають регулярно прибиратись вологим прибиранням, то цій конкретній палаті присвоюється властивість «має бути прибрана» (див. Рисунок 14).

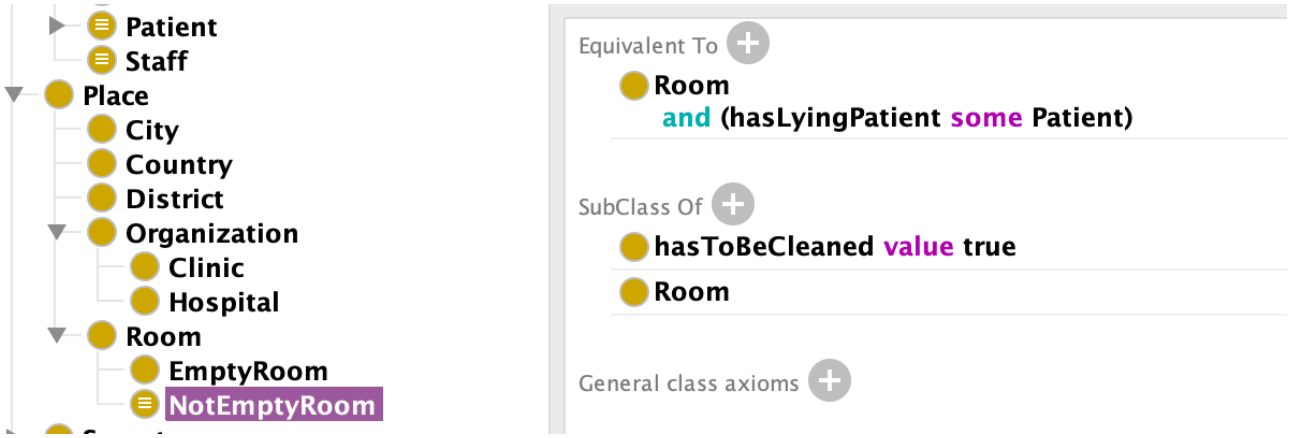


Рисунок 14 – Еквівалент і підклас Непустої палати

Додаток В ілюструє класи та відносини між ними у вигляді графу за допомогою плагіна OntoGraf.

2.3.4 Визначення правил логічного виведення

В онтологій також можна використовувати правила SWRL для того, аби задати певні правила, за якими система може робити логічні висновки і формувати нові знання. В моїй онтології я сформувала лише декілька правил, таких як:

а) Якщо пацієнт Б «займає ліжко» А і ліжко А «знаходиться в палаті» В, то пацієнт Б «має палату» В.

б) Якщо пацієнт А «має проблеми з диханням», «знаходиться в» лікарні Б і апарат штучної вентиляції В «знаходиться в» лікарні Б, то пацієнт А «має скористатись апаратом штучної вентиляції» В.

в) Якщо сімейний лікар А і лікар Б працюють в одній організації В, а лікар А «має контракт з пацієнтом» С, то сімейний лікар А «може бути заміненим на» сімейного лікаря Б.

2.3.5 Наповнення бази знань екземплярами

Останнім етапом – є створення окремих екземплярів класів у ієрархії. Я створила декількох лікарів та надала їх імя, прізвище, визначила в якій організації вони працюють та їхні робочі дні (Рисунок 15) . Також створила місто Київ, декілька районів, вказавши що вони знаходяться в Києві. Також створила декілька лікарень, вказавши райони, де вони знаходяться, і палати,

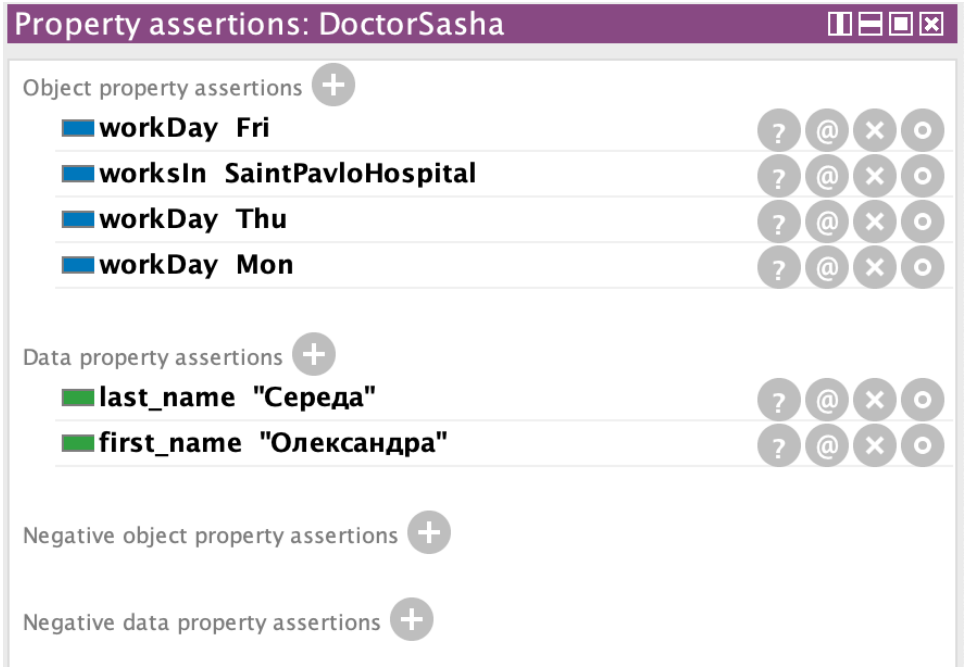


Рисунок 15–Екземпляр сімейного лікаря Олександра Середа

вказавши лікарні, де вони розташовуються. Потім я створила екзмепляри ліжок, надавши їм зв’язок з кімнатами, де вони стоять (Рисунок 16).

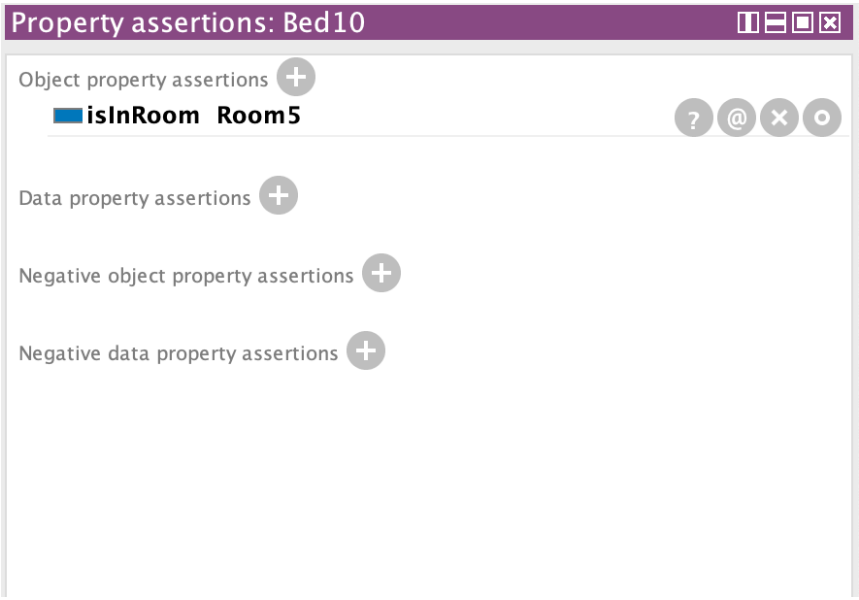


Рисунок 16 – Екзмепляр ліжка, що знаходиться в кімнаті 5

І, наостанок, створюємо екземпляри пацієнтів, надавши їм ім’я, прізвище, вказавши їх симптоми, а також з яким сімейним лікарем вони мають підписаний контракт (Рисунок 17).

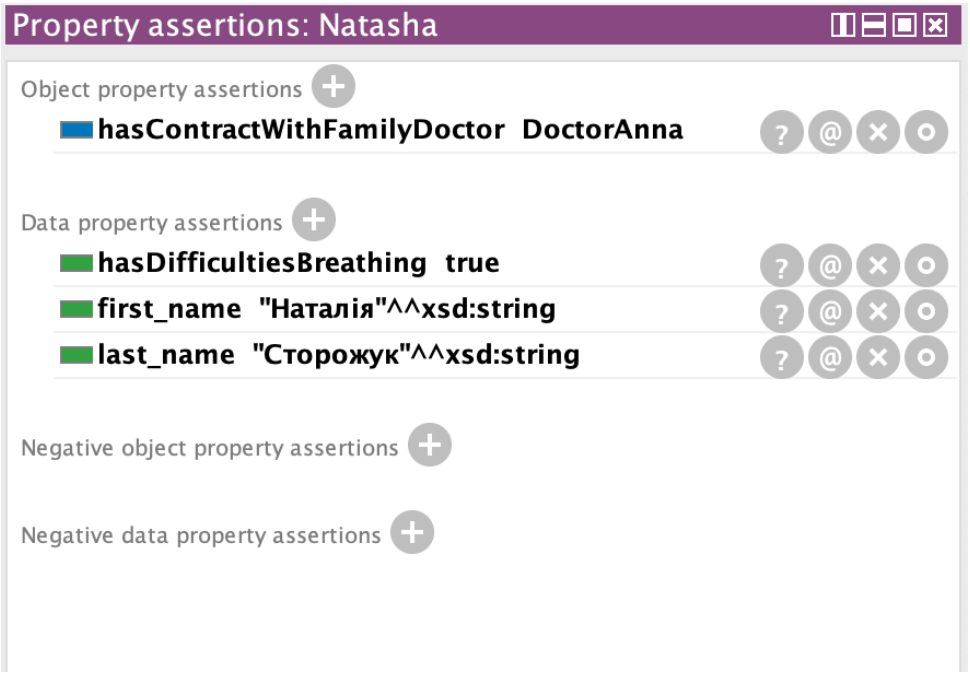


Рисунок 17– Екземпляр пацієнта Наташа Сторожук

2.4 Тестування розробленої бази знань

Як засіб тестування бази знань я обрала мову запитів до даних, представлених по моделі RDF, – SPARQL. Для виконання SPARQL-запитів буде використано плагін Protege – Snap SPARQL, що дозволяє отримувати дані, які були виведені різенором на основі типів зв'язків, обмежень онтології, а також застосування правил SWRL.

Почнемо з простих запитів. Необхідно вивести для кожного сімейного лікаря в кожній лікарні даної мережі кількість пацієнтів, з якими у цих лікарів є підписані договори. В базі знань ми початково вводили лише інформацію у кожного пацієнта – з яким лікарем вони мають договір, за допомогою зв'язку *hasContractWithFamilyDoctor*. Виконавши ж такий запит, ми ж отримуємо такі результати (Рисунок 18).

```
SELECT ?name ?lname ?hospital  (COUNT(?doctor) as ?numPatients)
WHERE
{
  ?doctor  :worksIn ?hospital .
  ?hospital :isLocatedIn :Kyiv .
  ?doctor  :hasContractWithPatient ?o .
  ?doctor :first_name ?name .
  ?doctor :last_name ?lname
}
GROUP BY ?name ?lname  ?hospital
```

Execute

?name	?lname	?hospital	?numPatients
Катерина^^xsd:string	Кобильська^^xsd:string	:SaintPavloHospital	3
Олександра^^xsd:string	Середа^^xsd:string	:SaintPavloHospital	1
Анна^^xsd:string	Петрова^^xsd:string	:SaintPetroHospital	1
Марія^^xsd:string	Шостка^^xsd:string	:SaintPetroHospital	1
Марія^^xsd:string	Котел^^xsd:string	:SaintPavloHospital	1

Рисунок 18 – Отримання кількості пацієнтів у кожного сімейного лікаря Києва

В результаті ми отримали кількість пацієнтів для кожного сімейного лікаря кожної лікарні, що знаходиться в Києві. Хоча ми початково явно не вказували в онтології, які лікарні знаходяться в Києві і які лікарі мають договори з якими пацієнтами. Ці висновки були зроблені на основі транзитивності зв'язку isLocatedIn та інверсивності зв'язків *hasContractWithFamilyDoctor* і *hasContractWithPatient*.

Далі ми хочемо отримати всі робочі дні сімейного лікаря, з яким даний пацієнт має підписаний контракт, та тих сімейних лікарів, які можуть замінити

його в разі відсутності на роботі. Виконуємо запит та отримуємо результати (Рисунок 19). В результаті ми отримали одного сімейного лікаря, який працює

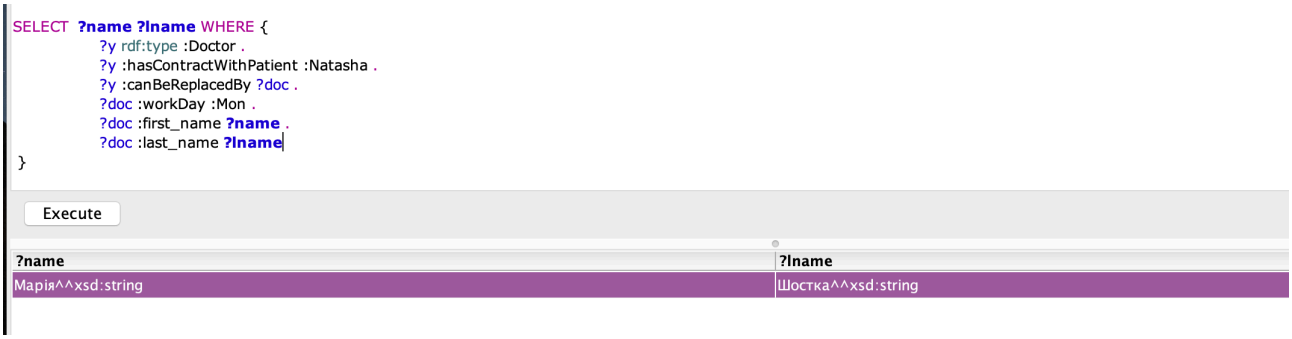


Рисунок 19 – Отримання лікарів, які в цей день можуть прийняти Наталію

по понеділках і може прийняти в цей день Наталію. Зазначимо, що в Наталії є контракт з сімейним лікарем Анною Петровою, проте в цей день вона не працює. Але за допомогою SWRL правила (див. пункт (в) в розділі 2.3.4) в онтології зроблено висновок, що Марія Шостка може замінити Анну Петрову. При цьому зв'язок *hasContractWithPatient* також явно не вказувався в онтології, а був виведений на основі інверсивності. Варто також вказати, що якби Анна Петрова працювала в цей день, то вона теж би вивелась в результатах такого запиту за рахунок рефлексивності зв'язку *canBeReplacedBy*.

Наступним запитом ми хочемо дізнатись, які ліжка та палати є вільними, щоб туди покласти нового пацієнта лікарні. Коли ми кладемо пацієнта на стаціонар, ми просто вказуємо ліжко, де він буде розміщений, тобто надаємо пацієнту зв'язок *occupiesBed* з якимось ліжком, який має зв'язок *isInRoom* з якоюсь кімнатою. Подальші висновки система робить самостійно, а саме виведення таких результатів при запиті на вільні місця (Рисунок 20). Зв'язок *hasBed* – інверсивний до *isInRoom*, тобто кожний раз коли ліжко отримає властивістю якусь кімнату, ця кімната після логічного виведення теж отримує свій зв'язок з цим ліжком. Клас OccupiedBed в онтології має еквівалент *Bed and isOccupiedBy some Patient*. Тобто цей клас надається екземпляру ліжка, якщо ліжко має зв'язок *isOccupiedBy* з певним пацієнтом, а цей зв'язок є інверсивним до зв'язку *occupiesBed*, який ми явно задаємо кожному пацієнту на стаціонарі.

Snap SPARQL Query:

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX : <http://www.semanticweb.org/annemanzhura/ontologies/2020/clinic#>

SELECT ?bed ?room
WHERE {
  { ?bed rdf:type :Bed .
   ?room :hasBed ?bed. }
  Minus {?bed rdf:type :OccupiedBed}
}
```

Execute

?bed	?room
:Bed12	:Room6
:Bed11	:Room5
:Bed10	:Room5
:Bed6	:Room2
:Bed15	:Room7
:Bed8	:Room4
:Bed7	:Room3
:Bed9	:Room5

Рисунок 20 – Отримання вільних ліжок і палат

Таким чином різонер формує нові твердження про те, які ліжка є зайнятими, і виводяться результатом усі вільні. Також таким способом ми можемо дізнатись скільки зайнятих ліжок є в лікарні, або в районі, або навіть в місті.

Скористаємось таким запитом (Рисунок 21). Як система визначає зайняті ліжка було пояснено вище, а як щодо розташування кожного ліжка і як з онтології вивелось, що вони всі знаходяться в певній лікарні і тим паче місті?

Snap SPARQL Query:

```
PREFIX owl: <http://www.w3.org/2002/07/owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX : <http://www.semanticweb.org/annemanzhura/ontologies/2020/clinic#>

SELECT ?hospital  (count(distinct ?bed) as ?count)
WHERE {
   ?bed rdf:type :OccupiedBed .
   ?hospital rdf:type :Hospital .
   ?bed :isLocatedIn ?hospital .
   ?bed :isLocatedIn :Kyiv
}
GROUP BY ?hospital
```

Execute

?hospital	?count
:SaintPavloHospital	5
:SaintPetroHospital	2

Рисунок 21– Отримання кількості зайнятих місць по лікарнях міста Києва

Подивитись логіку за цим твердженням можна в самому Protégé, натиснувши на автоматично сформований зв’язок *isLocatedIn* у якогось ліжка.

Отримаємо таке пояснення (Рисунок 22). Оскільки було явне твердження, що Room1 знаходиться в лікарні Святого Павла, а оскільки зв’язок є батьківською властивістю isInRoom, то вона передає їй всі свої характеристики, а саме в цьому випадку – транзитивність. Отже, отримуємо, що ліжко Bed1 знаходиться в лікарні Святого Павла, хоча ми це явно не вводили в онтологію. Такий самий алгоритм пояснення і з містом.

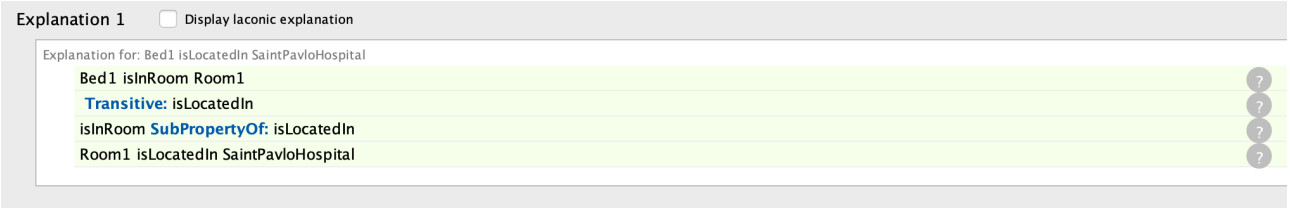


Рисунок 22– Пояснення формування нового зв'язку isLocatedIn у ліжка Bed1

Далі ми хочемо дізнатись скільки в лікарнях лежить пацієнтів з серйозними проблемами з диханням. Отримуємо ми це таким чином (Рисунок 23). Клас PatientWithBreathessness за визначенням в онтології надається тому екземпляру класу Пацієнт, що *((hasToUse some VentilationApparatus) or (uses some VentilationApparatus)) and (hasRoom some Room),* тобто такому, який лежить на стаціонарі і має за собою закріплене ліжко і який або використовує вже апарат штучного дихання, або система радить йому його використати. Яке вона це радить? За рахунок правила SWRL, який ми визначили на етапі проектування (див. пункт (б) розділу 2.3.4). Зв’язок *hospitalizedIn* формується

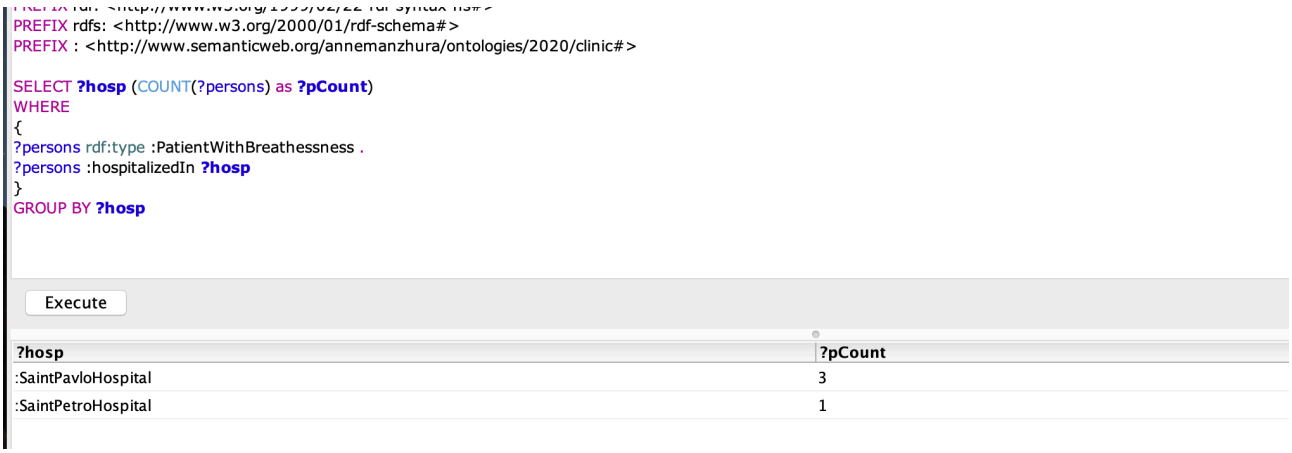


Рисунок 23–Отримання кількості пацієнтів з серйозним порушенням дихання

за рахунок логічного ланцюжка: якщо пацієнт має палату А, яка знаходиться в лікарні Б, то пацієнт госпіталізований в лікарні Б. Зв'язок «має палату» виводиться за рахунок правила SWRL (див. пункт (а) розділу 2.3.4).

Наступним запитом ми можемо дізнатись, які кімнати треба прибрати прибиральниці (Рисунок 24). Властивість *hasToBeCleaned* зі значення true за визначенням додається до усіх екземплярів класу NotEmptyRoom (Рисунок 25). Бо експерт і розробник бази знань робить припущення, що прибирати вологим прибиранням регулярно варто лише ті кімнати, де мешкають пацієнти. Клас NotEmptyRoom за визначенням надається тим палатам, що мають зв'язок *hasLyingPatient* з якимсь пацієнтом. А цей зв'язок є інверсійним до зв'язку *hasRoom*, який формується між пацієнтом і палатою внаслідок правила SWRL (див. пункт (а) розділу 2.3.4).

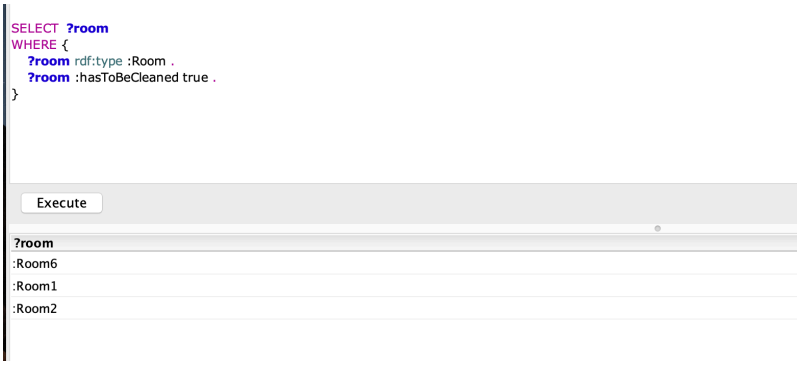


Рисунок 24–Виведення кімнат, які треба прибрати

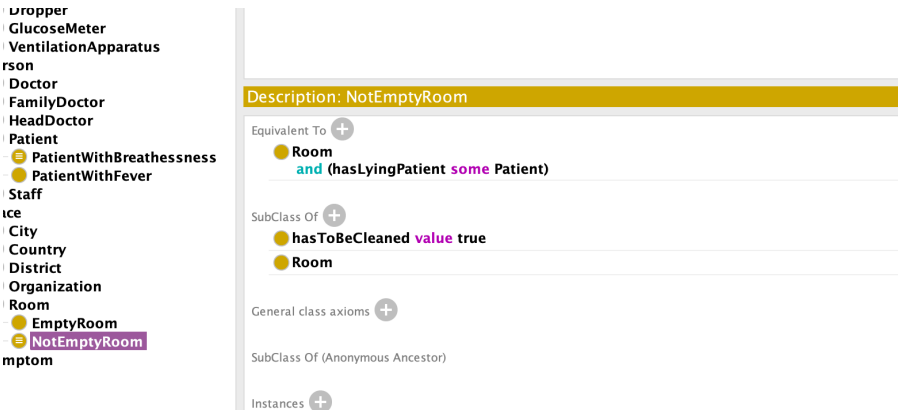


Рисунок 25–Визначення класу NotEmptyRoom

2.5 Подальша розробка

Розроблена мною база знань може бути вдосконалена шляхом надання їй зручного користувацького інтерфейсу у вигляді веб-додатку, до якого б мали доступ всі працівники медичного закладу. Такий веб-застосунок потребував би серверу, написаного на Java, який би використовував бібліотеки для роботи з онтологічними моделями в форматі OWL/RDF(S). Серед таких бібліотек варто назвати OWL API, Jena Apache та ONT API, які надають можливості написання SPARQL-запитів для отримання знань чи додавання нових тверджень в онтологію, а також підключення різонерів для логічного виведення знань, які початково не були явно закладені в базу.

Висновки

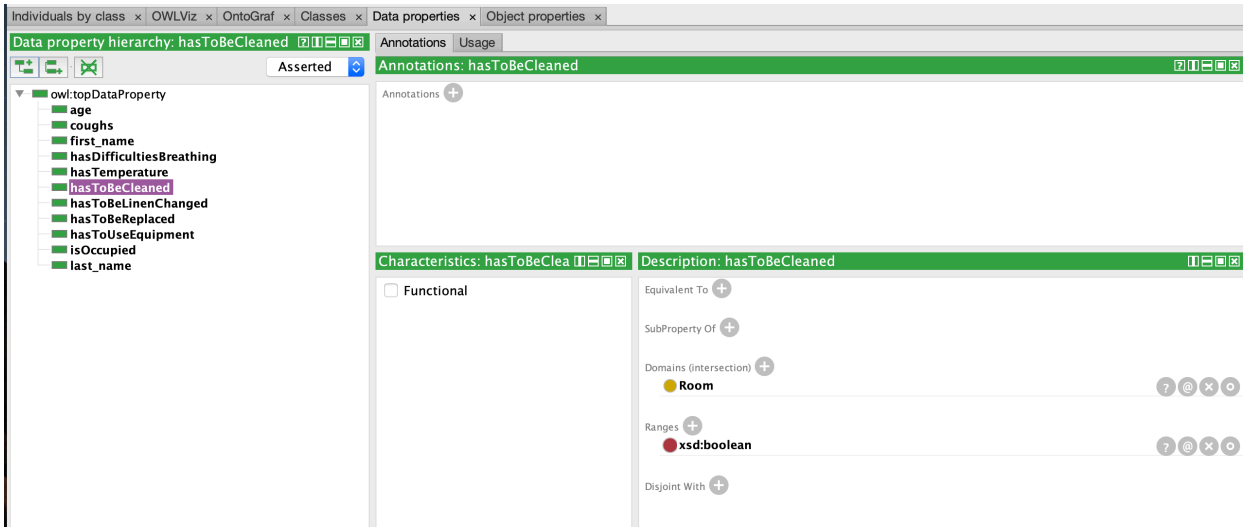
У цій роботі було досліджено ефективні способи та засоби розробки баз знань у прикладних корпоративних системах, а саме – онтологічне моделювання для представлення знань. Було наведено визначення поняття онтологія, пояснено її структуру, класифікацію та способи використання. Було покроково пояснено процес створення онтологічної моделі, чітко окреслені її задачі та цілі. Також було оглянуто та порівняно основні технології розробки онтологій, а саме мови опису, редактори, різонери. За результатом цього дослідження було обрано найбільш оптимальний редактор онтологій – Protégé, встановлено необхідні плагіни та різонер Pellet.

У практичній частині було розроблено онтологічну модель в Protégé, як основу для подальшої реалізації бази знань медичного закладу. Засобами специфікації SPARQL було продемонстровано переваги онтологічного представлення даних над більш традиційними, а саме можливість формування власних висновків на основі наявних тверджень та правил, таким чином виводячи нову інформацію і перетворюючи роботу з сирими даними на роботу із повноцінними знаннями.

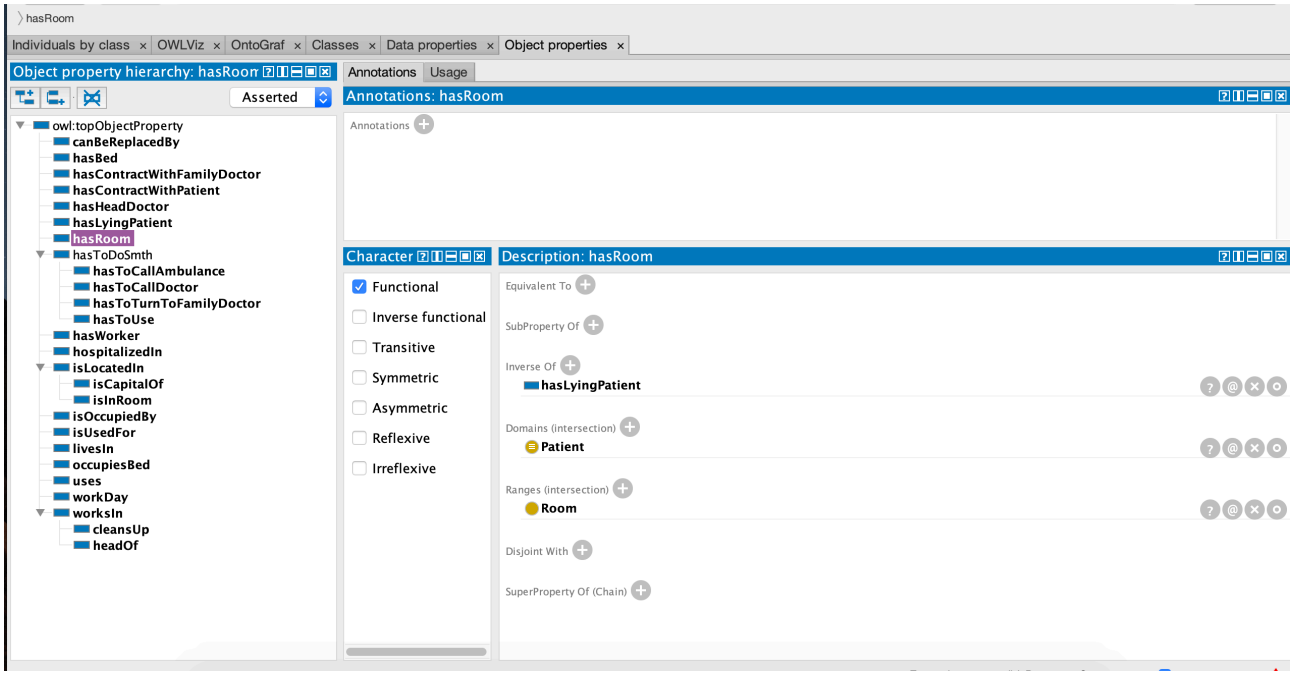
Список використаної літератури

1. https://stud.com.ua/45678/menedzhment/ontologiyi_klasifikatsiya
2. http://www.management.com.ua/ims/ims115.html
3. Gruber, T.R. (1993). A Translation Approach to Portable Ontology Specification.
4. http://www.ef.uns.ac.rs/mis/archive-pdf/2013%20-%20No2/MIS2013-2-4.pdf
5. https://www.researchgate.net/publication/221292703_Survey_on_Ontology_Languages
6. Морозов І., Белокопитова Є. «Анализ и сравнение работы различных Reasoner'ов в Protégé»
7. Горшков С. (2016). «Введение в онтологическое моделирование»
8. Лапшин В.А (2009). «Онтологии в информационных системах»

Додаток А (обов'язковий) Скріншот Protégé з усіма Data Properties



Додаток Б (обов’язковий) Скріншот Protégé з усіма Object Properties



Додаток В (обов'язковий) Скріншот Protégé з OntoGraf