МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА СПЕЦІАЛІСТА

на тему: **«РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ГРАФІЧНОГО АНАЛІЗУ ВІДПОВІДНОСТІ ОНТОЛОГІЙ ВИМОГАМ»**

Виконав: студент 1 курсу, групи 7.1226-з
спеціальності
122 Комп'ютерні науки та інформаційні технології
(шифр і назва спеціальності)
О.В. Василейко
(ініціали та прізвище)
Керівник доцент кафедри комп'ютерних наук, доцент, к.ф.-м.н. Єрмолаєв В.А.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)
Рецензент доцент кафедри програмної інженерії, к.т.н. Чопоров С.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2017

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

РЕФЕРАТ

Кваліфікаційна робота спеціаліста «Розробка програмного забезпечення для графічного аналізу відповідності онтологій вимогам»: 91 сторінка, 29 рисунків, 2 таблиці, 31 джерела, 9 додатків.

Об’єкт дослідження – аналіз та редагування UML діаграм з подальшою конвертацією у OWL онтології, згідно вимогам предметної області.

Мета роботи – розробка програмного продукту для конвертації UML діаграм в OWL онтології, з наданням можливості виконання попереднього графічного аналізу та редагування цих діаграм. А також графічний аналіз відповідності отриманих OWL онтологій вимогам.

Метод дослідження – описовий, порівняльний, алгоритмізація, програмування.

Апаратура – персональний комп’ютер, програмне забезпечення.

Результат дипломного проекту – програмний продукт який дозволяє аналізувати та редагувати UML діаграми. Конвертувати UML діаграми у OWL онтології, а також зберігати онтології у файл з розширенням .owl.

OWL, UML, XMI, RDF, RDF(S), XML, OMG, W3C, ARGOUML, ОНТОЛОГІЯ.

ABSTRACT

Specialist's qualifying paper «The Development of Software for Graphical Analysis of Ontology Fitness to Requirements»: 91 pages, 29 figures, 2 tables, 31 references, 9 supplements.

The object of the study – analysis and editing of UML diagrams with further conversion into OWL ontologies according to the required domain.

The aim of the study – software development for the UML diagrams conversion into the OWL ontologies, providing the possibility of previous graphical analysis and editing for these diagrams and graphical analysis of obtained ontology fitness to the requirements.

The methods of research – descriptive, comparative, algorithmic, programming.

The equipment – personal computer, software.

Result of graduation project – software which allows analyzing and editing of the UML diagrams, conversion of the UML diagrams into the OWL ontologies and ontologies saving as a file with .owl extension.

OWL, UML, XMI, RDF, RDF(S), XML, OMG, W3C, ARGOUML, ONTOLOGY.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

AI	Artificial Intelligence, Штучний інтелект
OWL	Web Ontology Language, Мова опису онтологій
ПЗ	Програмне забезпечення
XML	Extensible Markup Language
XMI	XML Metadata Interchange, XML обмін метаданими
RDF	Resource Description Framework, Середовище опису ресурсів
DL	Description logic, логіка опису
URI	Uniform Resource Identifier, Уніфікований ідентифікатор ресурсів
UML	Unified Modeling Language, Уніфікована мова моделювання
API	Application programming interface, Програмний інтерфейс додатку
JDK	Java Development Kit, Комплект розробника застосунків на мові Java
UI	User Interface, Користувацький інтерфейс

ВСТУП

Актуальність теми. Історія штучного інтелекту показує, що знання має вирішальне значення для інтелектуальних систем. У багатьох випадках чим краще описане знання тим важливішим воно буде для вирішення конкретного завдання, навіть використання найкращих та найдосконаліших алгоритмів не може зрівнятися по важливості з точним описом знань. Онтології покликані вирішити цю проблему, вони можуть допомогти розв’язувати завдання з розуміння людської мови, повторного використання існуючих данних, швидкого пошуку інформації тощо.

Мета цього дипломного проекту полягає у створенні інструменту для виконання аналізу та редагування UML діаграм з подальшою можливістю конвертації у фомат OWL (Ontology Web Language) – мови онтології. А також виконання графічного аналізу відповідності отриманих OWL онтологій вимогам.

Розробка програмного продукту буде побудована на роботі з UML діаграмами. Тому діаграми UML виступають у ролі об’єкта цієї дипломної роботи, а предметом дослідження є конвертація UML у OWL, а також аналіз отриманих UML діаграм на основі ПЗ з дипломної роботи [1].

Методи дослідження – застосовувалися такі методи як: метод експертних оцінок, метод аналізу та порівняння, метод написання коду і його відлагоджування, методи проб та помилок.

Основними матеріалами цього дипломного проекту є роботи [2] та [3]. Згідно наданої інформації був розроблений OWL конвертор для програмного продукту ArgoUML.

Практичне значення цієї дипломної роботи можно висловити як наступний ланцюжок дій: використання результатів дипломного проекту [1] а саме отримання та імпорт UML діаграм з тексту природною мовою, проведення аналізу та редагування UML діаграм з наступною конвертацією

поточних даних у формат мови онтології OWL з наступним графічним аналізом відповідності отриманих онтологій вимогам. Ця послідовність реалізує можливість створювати та наповняти онтології даними.

Тема онтологій є дуже цікавою та неординарною для мене. Окрім того вона має безпосередній зв’язок з напрямком AI (Artificial Intelligence) у інформатиці, який також мене цікавить. Тому виконуючи цей дипломний проект, я хотів би детально ознайомитись з процесом створення, аналізу та редагування онтологій, а також зробити невеличкий внесок у цю галузь.

1 ОГЛЯД ОНТОЛОГІЙ І UML ДІАГРАМ, ІНСТРУМЕНТИ ДЛЯ СТВОРЕННЯ, РЕДАГУВАННЯ ТА АНАЛІЗУ

У цьому розділі я розгляну основні поняття онтологій, їх роль та сфери використання. Також я планую розповісти про інструменти які дають можливість проводити створення, аналіз та редагування онтологій. Більш за все мене цікавить питання мов онтологій. У цій темі я намагатимуся надати якомога більше інформації про онтології, описати їх особливості та навести приклади.

Далі йдеться про існуючі підходи аналізу та відображення різниці між онтологіями. Деякі з них я планую запозичити для реалізації свого дипломного проекту.

Найважливішою темою цього розділу будуть UML діаграми. Насамперед, потрібно ознайомитись з теоретичними даними та їх невід'ємною участю у реалізації ідеї проведення аналізу та редагування розпізнаного тексту дипломний проект [1].

І остання, але не меньш важлива частина, буде присвячена редактору ArgoUML як основному інструменту аналізу та редагування UML діаграм.

1.1 Онтології

1.1.1 Визначення

Термін "онтологія" може бути визначений як явна специфікація концептуалізації. Онтологія захоплює структуру домену, тобто концептуалізації. Це включає в себе модель домену з можливими обмеженнями. Концептуалізації описують знання про предметну область, а не

про конкретний стан справ в області. Іншими словами, концептуальне не змінюється, або змінюється дуже рідко [4].

Онтологія – це спроба всеосяжної і докладної формалізації деякої області знань за допомогою концептуальної схеми.

Онтологія, являє собою спробу найбільш універсального опису існуючого, який не обмежувався б даними окремих наук і, можливо, не зводився б до них.

На даний момент у терміна онтологія існує два значення: онтологія у філософії та онтологія у комп’ютерній науці. В нашому випадку ми розглядаємо другий варіант.

Зазвичай така схема складається зі структури даних, що містить всі релевантні класи об'єктів, їх зв'язку і правила (теореми, обмеження), прийняті в цій галузі. Цей термін в інформатиці є похідним від стародавнього філософського поняття «онтологія».

Як правило, онтології складаються з наступних елементів: поняття (класи), індивіди (екземпляри), відносини і атрибути.

Індивиди (екземпляри) – це основні компоненти онтології на нижчому рівні. Індивіди можуть являти собою як фізичні об'єкти (люди, будинок, планета) так і абстрактні (числа, слова). Однією з головних цілей онтології є класифікація індивідів.

Поняття (класи) – абстрактні групи, колекції або набори об'єктів. Вони можуть включати в себе екземпляри, інші класи, або ж поєднання і того, й іншого [5].

Об'єкти в онтології можуть мати атрибути. Кожен атрибут має, принаймні, ім'я і значення, і використовується для зберігання інформації, яка специфічна для об'єкта і прив'язана до нього.

Відносини – зазвичай ставленням є атрибут, значенням якого є інший об'єкт.

1.1.2 Типи та призначення

Онтології класифікуються, як онтології різних засобів з використанням таких критеріїв, як ступінь абстракції і область застосування:

Upper ontology – концепції, що підтримують розвиток онтології, мета-онтології.

Domain ontology – поняття, які стосуються конкретної теми або області, що представляє інтерес, наприклад, в області інформаційних технологій або комп'ютерних мов або окремих галузей науки.

Interface ontology – концепти, які мають відношення до межі двох дисциплін.

Process ontology – входи, виходи, обмеження, секвентування інформації, бере участь в ділових або технологічних процесах.

У даний час онтології використовуються в процесі програмування як форма представлення знань про реальний світ або його частини. Основними сферами застосування є: моделювання бізнес-процесів, семантична павутина (Semantic Web) та штучний інтелект (AI).

Найчастіше онтології використовують:

а) з метою спільного використання людьми або програмними агентами загального розуміння структури інформації;

б) можливості повторного використання знань в предметній області;

в) для того щоб зробити допущення в предметній області явними;

г) для відділення знань в предметній області від оперативних знань;

д) для аналізу знань в предметній області.

1.1.3 Мови опису онтологій

Існує безліч мов навколо онтології, перш за все це Web Ontology Language OWL. Як виклали у статті [6], ця мова була розроблена на основі досвіду його попередника DAML+OIL, а дизайн OWL був виконаний у робочих групах W3C. OWL2 є розширенням OWL і є рекомендацією W3C.

OWL називається Web Ontology Language. Він заснований на веб-стандартах, таких як XML, IRIs та RDF, а також розроблений таким чином, що може бути використаний через Інтернет [6].

OWL файл має можливість імпортувати інші об’єкти використовуючи їх URI.

<owl:Class rdf:about="http://www.w3.org/2002/07/owl# Cat"/>

Мова OWL має нову версію OWL2. Вона була створена як результат відгуків користувачів і зокрема, інформації, зібраної в ході проведення семінарів. Окрім того що OWL2 включає в себе більш вдосконалені базові функції, вона також має великий перелік нових функцій з широким спектром використання. Саме вони допомагають покращити роботу маніпулювання даними та виконати детальніший опис конкретної теми.

1.1.4 OWL, RDF, RDFS

Ці мови включають у собі велику кількість опцій і варіантів. OWL поставляється у трьох варіантах (OWL Full, OWL Lite і OWL DL). OWL2 поставляється з двома семантичними підходами (тобто два способи визначення сенсу онтології, прямий і на основі RDF) і трьох профілів (тобто фрагменти або синтаксичні обмеження, так звані OWL2 EL, QL та RL) [6]. Тому є можливість вибрати між кількома варіантами щоб зберегти онтологію.

OWL2 Functional Syntax – Визначає запис онтології у вигляді функції або набору функцій.

Ontology(<http://example.com/tea.owl> Declaration(Class(:Tea)))

OWL2 XML Syntax – Визначає XML серіалізацію, що моделює структуру онтології OWL2 [7].

```
<Ontology ontologyIRI="http://example.com/tea.owl" ...>
  <Prefix name="owl" IRI="http://www.w3.org/2002/07/owl#"/>
  <Declaration>
    <Class IRI="Tea"/>
  </Declaration>
</Ontology>
```

Manchester Syntax – Являє собою компактний, легкий для читання синтаксис. Являє собою синтаксис на основі кадрів (frame-based) на відміну від інших типів синтаксису які використовують аксіоми (axiom-based). Варіації доступні для OWL і OWL2. Але не всі онтології OWL і OWL2 можуть бути виражені за допомогою цього типу синтаксису [7].

Ontology: <http://example.com/tea.owl> Class: Tea

RDF/XML Syntax – RDF/XML є нормативним для сім'ї мов OWL. Він містить декілька форматів які доступні для використання [7].

```
<rdf:RDF ...>
    <owl:Ontology rdf:about=""/>
    <owl:Class rdf:about="#Tea"/>
</rdf:RDF>
```

RDF/Turtle – дозволяє записати RDF граф у компактній формі, з використанням природної мови із скороченнями для загальних моделей використання та типів даних [7].

<http://example.com/tea.owl>
rdf:type owl:Ontology. :Tea rdf:type owl:Class

1.2 Огляд сучасних інструментів для роботи з онтологіями

1.2.1 Інструменти, порівняльні характеристики

Нижче надані відомості про існуючі інструменти для роботи з онтологіями, а також надана таблиця з порівняльними характеристиками для таких інструментів як Protégé, OntoStudio, Apollo, Swoop та TopBraid Composer Free Edition.

1.2.2 Protégé як найбільш відомий і використовуваний інструмент

Protégé є вільним редактором онтологій з відкритим кодом. Платформа Protégé підтримує два основних способи моделювання онтологій за допомогою редакторів Protégé-Frames і Protégé-OWL. Онтології у Protégé можуть бути експортовані у різні формати, наприклад RDF, RDFS, OWL та XML-схеми [9].

Protégé написаний на Java, є розширюваним та забезпечує plug-and-play підхід, що робить його гнучкою базою для швидкого прототипування і розробки додатків. Прикладами є візуальний редактор для OWL, який може бути використаний для редагування онтологій [10].

Як і у більшості інструментів моделювання, архітектура Protégé чітко розділяється на “model” та “view”. Model - це внутрішнє уявлення механізму онтологій і баз знань. View компоненти забезпечують інтерфейс для відображення базової моделі [12].

1.3 Існуючі підходи для відображення різниці між онтологіями

1.3.1 Теоретичні відомості

Графічний аналіз являє собою графічне зображення даних за допомогою діаграм, малюнків та графіків. Цей тип аналізу широко використовується під час презентації великих обсягів даних. Такий вид аналізу досить легкий для розуміння та грає велику роль у таких сферах як економіка, інженерія, програмування та інші.

У теорії використовують такі поняття як ontology mapping, ontology alignment або ontology matching, що являють собою процес визначення відповідності між поняттями в онтології.

Поняття ontology matching і ontology mapping диференціюються, в результаті чого перший відноситься до ідентифікації кандидата порівняння між онтологіями, в той час як другий відноситься до встановлення фактичних відповідностей між ресурсами онтологій на основі кандидатів порівняння [13].

1.3.2 Приклади існуючих рішень

Онтології розглядаються у якості будівельних блоків семантичної мережі, а разом з ними постає питання про сумісність даних.

Один із засобів досягнення семантичної сумісності здійснюється за допомогою крос-мовного (cross-lingual) відображення онтологій. Методи

перекладу часто використовуються як проміжний крок, щоб перевести концептуальні вирази в онтології. Такий підхід, по суті, знімає бар'єр природної мови у середовищі відображення і робить можливим застосування одномовних інструментів перетворення онтологій.

Підходи alignment paradigms та similarity-based також є досить потужними й гнучкими для виправлення онтологій виражених в таких мовах, як OWL. Визначена універсальна міра для порівняння сутностей двох онтологій, яка заснована на простому і однорідному принципі порівняння: Подібність залежить від типу об'єкта і включає в себе всі функції, які його визначають (наприклад, суперклас, властивості екземплярів та інші). Один-до-багатьох і приблизність у описі сутностей є ключовими труднощами в цьому контексті. Вони розглядаються через локальні зпівставлення множин сутностей і ітераційного обчислення рекурсивно залежних подібностей [13].

1.3.3 Protégé prompt plugin

Protégé Prompt plugin є дуже потужним додатком, який був розроблений для надання користувачу можливостей автоматично аналізувати і порівнювати онтології та відображати їх різницю у графічному вигляді [9].

Можливості PROMPT:

а) порівняння версій у одній онтології;

б) мапінг (перетворення) однієї онтології до другої;

в) переміщення рамок між включеними проектами;

г) поєднання двох онтологій в одну;

д) вилучення частин з онтології.

Проте Protégé Prompt не може повноцінно проводити аналіз і надавати гарантований результат. Основною причиною є те, що він не має можливості розуміти і розбирати контексти на рівні людини. Отже вагомим недоліком постає те, що плагін потребує постійного втручання користувача у процес

аналізу онтології та проведення його корекції. Але навіть цей факт не може приховати того що цей плагін вносить достатній вклад у полегшення машинного аналізу та трансформації онтологій.

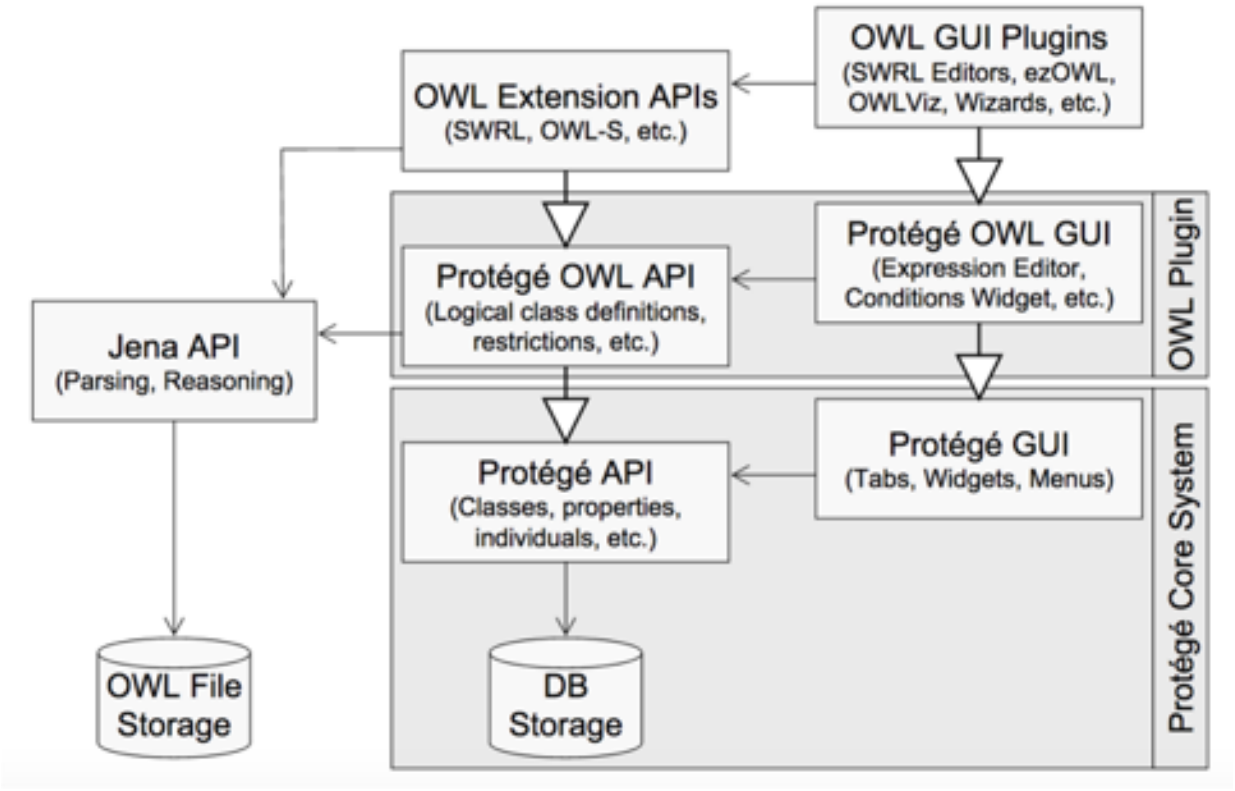


Рисунок 1.1 – Архітектура Prompt плагіна для Protégé [11]

1.4 Unified Modeling Language та UML діаграми

1.4.1 Опис мови UML

Unified Modelling Language – перекладається як універсальна мова моделювання. UML є мовою позначень та побудови діаграм. UML призначена для визначення, візуалізації і документування моделей, їх властивостей та залежностей [14]. Перш за все ця мова була розроблена для полегшення і прискорення роботи розробників ПЗ. Використовуючи UML діаграми можно

детально описати і відобразити архітектуру проекту, його модулі, а також їх взаємодію між собою.

Розробкою UML керує Object Management Group (OMG). Ця мова є загальноприйнятим стандартом графічного опису програмного забезпечення.

1.4.2 Види UML діаграм

Взагалі існує три класифікації UML діаграм:

Діаграми поведінки – діаграма, яка зображує поведінкові особливості системного процесу або бізнесу.

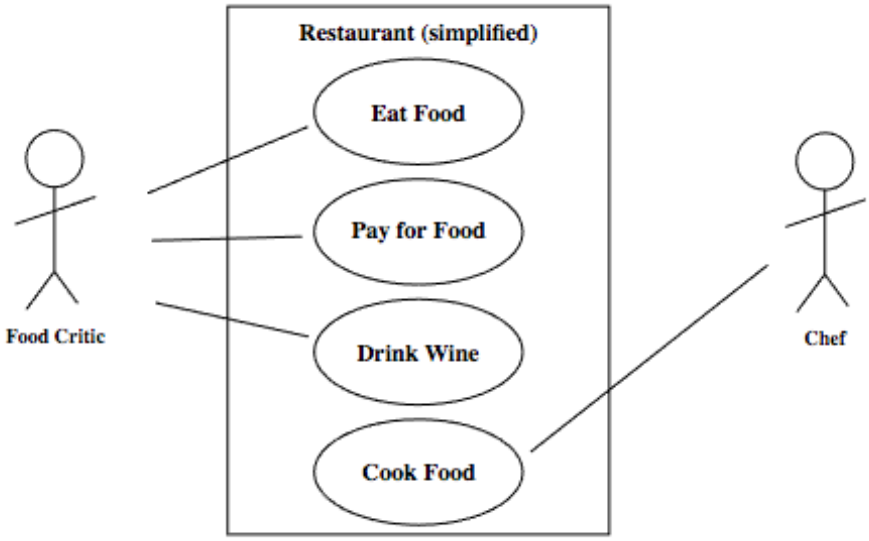


Рисунок 1.2 – Діаграма поведінки (Use case diagram)

Діаграми взаємодії – вони є підмножиною поведінкових діаграм, які підкреслюють взаємодію між об'єктами. Включають також в себе опис зв'язків, огляд взаємодії, послідовності та тимчасових діаграм.

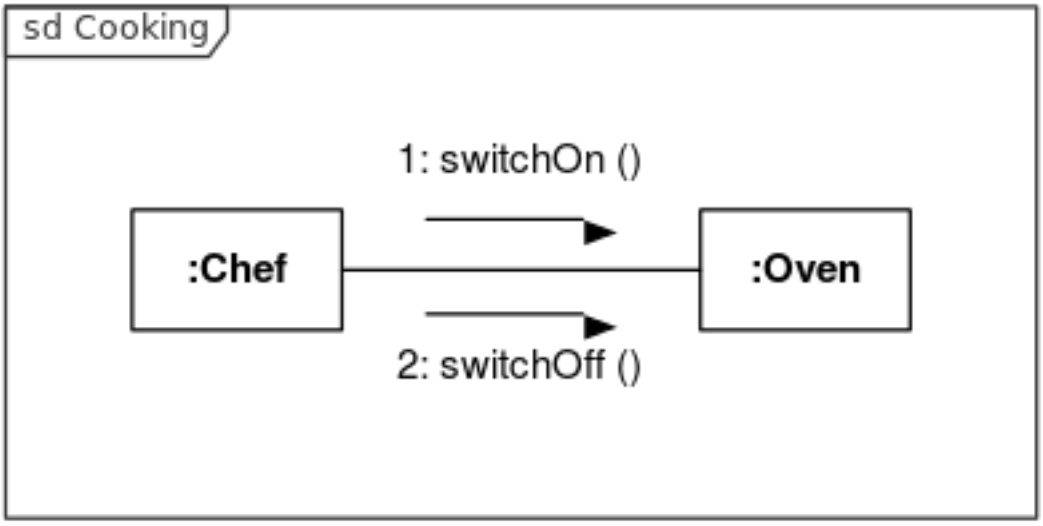


Рисунок 1.3 – Діаграма взаємодії (Communication diagram)

Структурні схеми – Тип діаграм, який зображує елементи специфікації. Зокрема зображують класи, композитні структури, компоненти, розгортання, об'єкти, та діаграми пакетів.

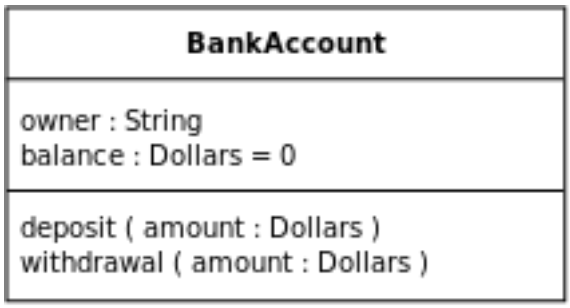


Рисунок 1.4 – Структурна діаграма (Class diagram)

1.4.3 Отримання результатів парсингу тексту у вигляді діаграми класів

UML діаграми у даному проекті використовуються як основний засіб передачі інформації та її конвертації у інші формати. Задача цього дипломного проекту складається у тому щоб отримати текст природною мовою у вигляді UML діаграми [1] та надати змогу користувачу провести аналіз. У випадку виявлення будь яких помилок, внести корективи, та зберегти діаграму у форматі OWL (як онтологію).

1.5 Огляд існуючих інструментів для роботи з UML діаграмами

1.5.1 Короткий перелік найбільш популярних редакторів на сьогодні

На сьогоднішній день створена велика кількість UML інструментів серед яких можна виділити найпоширеніші: ArgoUML, NetBeans, Microsoft Visio, Papyrus та Visual Paradigm for UML. Всі вони мають як переваги так і недоліки які детально описані у наступному пункті.

1.5.2 Порівняльні характеристики

Таблиця 1.2 – Порівняння інструментів для роботи з UML [15]

	Платформа	XMI	MDA(model driven architecture)	Мова програмування	Open Source
ArgoUml	Cross-platform (Java)	Так	-	C++, C#, Java, PHP4, PHP5, Ruby	Так
NetBeans	Windows, macOS, Linux, Unix	-	-	Java	Так
Microsoft Visual Studio	Windows	Plugin	-	-	Hi
Papyrus	Windows, Linux, macOS (Java)	Так	-	Ada 2005, C/C++, Java addins	Так
Visual Paradigm for UML	Cross-platform (Java)	комерційна версія	-	Java, C#, C++, PHP, Ada, Action Script (всі у комерційній версії)	Hi

1.5.3 ArgoUML і причина вибору

ArgoUML – інструмент для моделювання UML діаграм, який забезпечує підтримку об'єктно-орієнтованого проектування. Однією з основних причин вибору ArgoUML є те що він існує у вільному доступі, отже це дає змогу використовувати та розширювати функціонал цієї програми легально. Не менш важливим фактором є крос-платформна підтримка для таких ОС як MacOS, Windows, Linux, а також доступність програми на десяти мовах.

ArgoUML має добре продуманий графічний інтерфейс. Він допомагає проектувальникам в процесі прийняття рішень, шляхом забезпечення візуалізації конструктивних схем і простих синтаксичних перевірок. Крім того, інструменти ArgoUML забезпечують значні переваги у області контролю версій і супутніх механізмів проектування [16].

1.6 Висновки

У цьому розділі було розглянуто мови онтологій, такі як OWL та RDF, інструменти для роботи з ними, а також була нада порівняльна характеристика цих інструментів. Наступним була розглянута мова UML та популярні редактори які дозволяють створювати та редагувати UML діаграми. Також була надана порівняльна характеристика для цих редакторів. Наприкінці був більш детально описаний редактор ArgoUML, який буде використовуватися у цьому проекті як базовий. Набуті теоретичні знання будуть використані для створення механізму конвертації UML в OWL у наступних розділах, а також для змоги виконувати графічний аналіз онтологій у відповідності до вимог предметної області.

2 РОЗШИРЕННЯ ГРАФІЧНОГО РЕДАКТОРА ARGOUML, ВПРОВАДЖЕННЯ МЕХАНІЗМУ КОНВЕРТАЦІЇ UML В OWL

Так як одною з цілей цього дипломного проекту є надання можливості переглядати, аналізувати, редагувати UML діаграми, з подальшою конвертацією у OWL згідно вимогам PSI [2], мій вибір зупинився на розширенні редактора ArgoUML.

Окрім того що ArgoUML надає великі можливості для проведення аналізу та редагування UML діаграм, цей редактор знаходиться у відкритому доступі. Цей факт гарантує, що розробники та дослідники зможуть використовувати створений у цьому дипломному проекті код на вільній і легальній основі. Це може надати необхідний стимул для розвитку цього напрямку та допоможе отримати більш функціональну версії ArgoUML для кінцевих користувачів.

У процесі дослідження коду ArgoUML виявилося що він складається з більш ніж 20 внутрішніх проектів які пов’язані між собою. Це означало що його розширення не буде тривіальною задачею без попереднього дослідження всіх його складових частин.

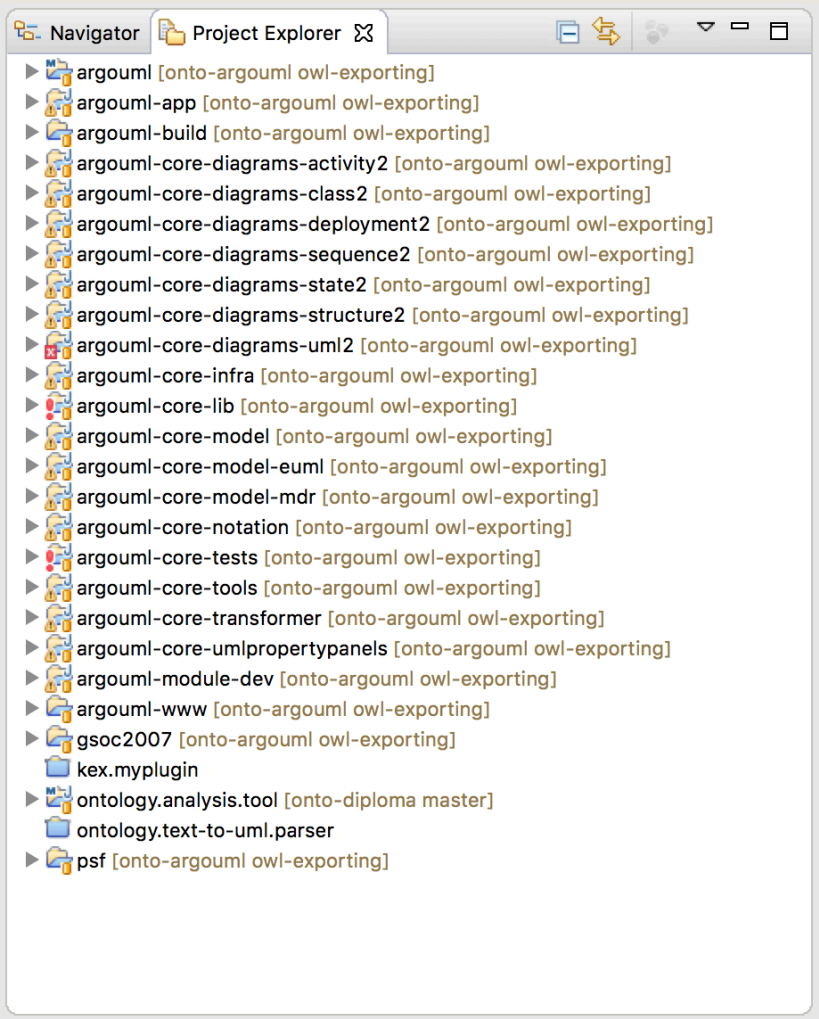


Рисунок 2.1 – Структура проектів ArgoUML (Project Explorer)

2.1 Опис розширення ArgoUML з урахуванням особливостей редактора

2.1.1 Дослідження коду і пошук можливостей розширення редактора

Перший етап проект xmi2owl – Одним із перших рішень для OWL конвертації була розробка окремого xmi2owl проекту який не пов’язаний з ArgoUML. В основі цієї ідеї було отримання XML файлу з даними після обробки діаграми в ArgoUML. Потім після розпізнання елементів XML файлу та зворотньої конвертації елементів у класи та залежності, цей модуль повинен

був виконати конвертацію у формат OWL. Щоб прискорити реалізацію було вирішено взяти за основу пакет класів XMIReader з відкритим кодом, розроблений компанією Google, та провести зміни відповідно до контексту цього дипломного проекту.

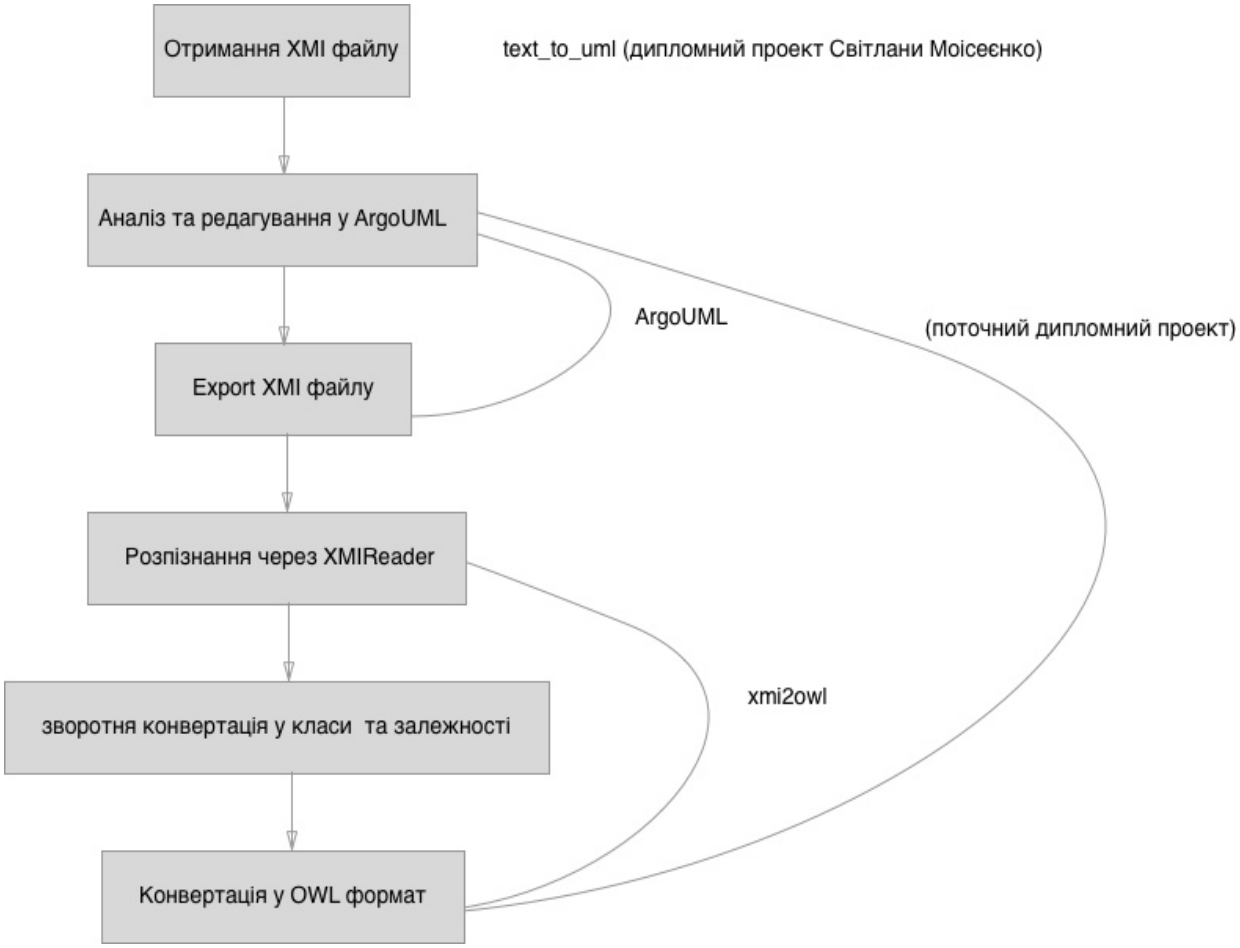


Рисунок 2.2 – Схема роботи проекту xmi2owl

Але ця ідея була хибка та додавала багато складнощів у її реалізацію, наприклад, XMI файл згенерований після редагування у ArgoUML буде мати багато відмінностей, між файлом який був створений за допомогою програми парсера з дипломної роботи [1]. Це означає наступне – нам буде потрібен ще один модуль конвертації для підготовки універсального XMI файлу. Наприклад, ПЗ Світлани генерувало файл з такими атрибутами

```
<UML:Model xmi.id="UML_model_1" name="NewModelUML"
        isSpecification="false" isRoot="false" isLeaf="false"
        isAbstract=“false”>
```

а результат зберігання XMI у ArgoUML включав у собі додатковий атрибут

```
<UML:Model xmi:type="uml:Model" xmi.id="UML_model_1"
        name="NewModelUML"
        isSpecification="false" isRoot="false" isLeaf="false"
        isAbstract=“false”>
```

Спочатку було вирішено внести необхідні корективи у роботу Світлани, таким чином щоб елементи були подібні у найбільш критичних для конвертації місцях, та зробити процеси створення XMI максимально однаковими.

```
public class AbstractModelElement implements Serializable {
        - @XmlAttribute (name="xmi:type")
        - public String _xmi_type;
        @XmlAttribute (name="xmi.id")
        public String _model_id;
        . . .
}
```

Але детальніше проаналізувавши можливості розвитку такої системи, було вирішено відмовитися від цієї ідеї. По-перше ризики помилок зростали разом із збільшенням вірогідності генерації відмінних атрибутів у наших проектах. По-друге такий підхід не є універсальним, тобто якщо ми отримаємо

XMI файл з будь-якого іншого джерела існує дуже велика ймовірність того що цей файл не буде конвертовано правильно.

Іншим рішенням було отримання UML даних беспосередньо у коді ArgoUML без попереднього створення XMI файлу. Цей підхід здавався більш надійним та гнучким. Тобто отримавши необхідні дані ми можемо проводити будь-яку подальшу конвертацію незважаючи на різні формати у XMI файлах.

Другий етап: розширеня ArgoUML – Основною задачею на цьому етапі було вивчання коду ArgoUML з наступним виділенням модулів які потрібні для розширення цього редактора. Такими модулями (проектами) стали argouml-app та argouml-core-model-mdr.

У модулі argouml-core-model-mdr були знайдені такі класи як XmiWriterMDRImpl та XmiReaderImpl (реалізує інтерфейс XmiReader та XmiExtensionWriter). На їх основі було вирішено, побудувати класи для отримання необхідних елементів UML діаграми та конвертації в OWL.

У модулі argouml-app були знайдені файли які відповідають за розширення інтерфейсу GenericArgoMenuBar та ActionExportXMI. Отже тут був створений клас ActionExportOWL який відповідає за дії при натисканні на меню File –> Extort to OWL…

Також у модулі argouml-app знайдено клас XmiFilePersister (він реалізує інтерфейс AbstractFilePersister) і відповідає за процес конвертації у XMI формат. Отже наступним був побудований схожий за властивостями клас OWLFilePersister для виконання конвертації в OWL.

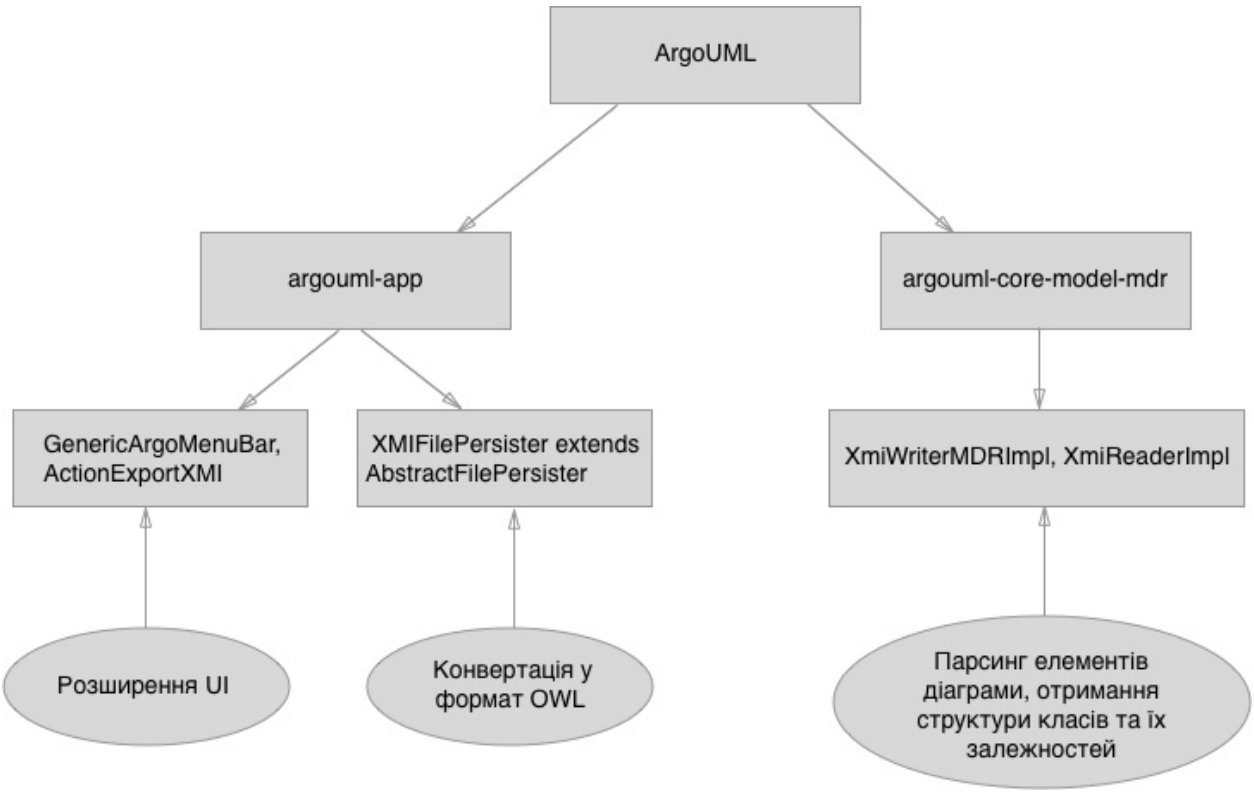


Рисунок 2.3 – Схема розширення ArgoUML

Незважаючи на те що цей підхід включає проведення додаткових змін не тільки у файлах які описані вище, що становить додаткову складність для отримання фінального результату. Цей підхід на даному етапі виявляється більш надійним та гнучким на відміну від перших спроб з проектом xmi2owl. Тому було вирішено вибрати саме його.

2.1.2 Пошук відповідної бібліотеки для конвертації UML діаграми в OWL формат

На цьому етапі дослідження було проведено порівняння існуючих бібліотек та фреймворків які спроможні виконувати конвертацію класів, їх атрибутів та залежностей у формат OWL Web Ontology Language (мову онтологій). Основним критерієм вибору була можливість додавання та взаємодії з кодом ArgoUML. В результаті цього дослідження було прийняте рішення використовувати фреймворк Apache Jena, який має широкий набір можливостей та прекрасно розуміється з ArgoUML, а також знаходиться у відкритому доступі.

2.1.3 Apache Jena як засіб конвертації у OWL

Jena є фреймворком для побудови Semantic Web Applications. Забезпечує програмне середовище для роботи з форматами RDF, RDFS та OWL, а також включає в собі механізм логічного висновку, заснований на заздалегідь заданих правилах [17]. Бібліотека надає API (прикладний програмний інтерфейс) для проведення конвертації класів, їх властивостей та залежностей у різні фомати мови онтологій.

Jena підтримує серіалізацію RDF графів у:

а) реляційну базу даних;

б) RDF/XML;

в) OWL ;

г) Turtle;

д) Notation 3 (non-XML серіалізація).

2.1.4 Пошук шляху розширення з урахуванням поточної архітектури та дотриманням методичних рекомендацій ArgoUML

Враховуючи той факт що ArgoUML виявляється масштабним проектом який складається з багатьох інших проектів, у вигляді окремих модулів, досить складною задачею виявилось дослідження коду з метою проведення його розширення та дотримання цілісності існуючої архітектури.

По-перше була знайдена можливість отримання інформації щодо побудови поточної діаграми UML та доступ до її елементів, точніше отримання класів та їх атрибутів.

Згодом були отримані наступні дані, такі як залежності типу асоціація, композиція, агрегація та генералізація.

Згодом у процесі подальшого дослідження коду була отримана інформація про наступні типи залежностей: one-to-one, zero-to-one, one-to-many, many-to-many.

Важливим етапом виявився пошук модулю для впровадження можливості зберігати дані у вигляді файлу.

На базі отриманих результатів був створений класс OwlWriterMDRImpl та інтерфейс OwlWriter. Класс OwlWriterMDRImpl включає в себе весь необхідний функціонал для отримання даних з поточної діаграми, проведення попереднього аналізу її елементів, та конвертацію у формат мови онтологій OWL, а також запис у файл формату .owl. OwlWriter у свою чергу надає необхідний інтерфейс для запису та конвертації UML даних в OWL. OwlWriterMDRImpl імплементує інтерфейс OwlWriter.

На фінальному етапі проведено дослідження можливості розширення користувацького інтерфейсу, а саме панелі меню та впровадження классу ActionExportOWL на основі базового класу AbstractAction.

Усі класи та інтерфейси яки були створені вище, відповідають нормам та методичним рекомендаціям ArgoUML. Також було дотримано

стилістичного форматування коду та опису дій у коментарях які використовується у переважній більшості коду ArgoUML.

2.1.5 Опис проблем при розширенні ArgoUML і спробі впровадження 3rd party бібліотек

ArgoUML був задуманий як інструмент і середовище для аналізу та проектування об'єктно-орієнтованих програмних систем [16].

Після детального вивчення проекту та його особливостей з метою реалізації можливості виконання конвертації UML об’єктів в OWL формат, необхідно було провести впровадження бібліотеки Apache Jena. На перший погляд це здавалось легким завданням, але на практиці виявилося що це не зовсім так. У процесі впровадження бібліотеки Jena виявилось, що вона потребує JDK 1.8 і це не співпадає з версією яку ArgoUML потребує для компіляції (JDK 1.6). Це означає що ми не можемо побудувати ArgoUML з цією бібліотекою не змінивши один з проектів. У результаті цього було вирішено змінити версію компілятору у деяких модулях ArgoUML до JDK 1.8. У кількох модулях це було досить складно зробити. Тому було вирішено залишити попередню версію JDK для окремих модулів та провести зміни у класах де передбачалось використання Apache Jena. Враховуючи те що розробка дипломного проекту проводилася на операційній системі OSX також довелося видалити з Eclipse JDK 1.6. і залишити тільки JDK 1.8. тому що операційна система OSX використовує системну версію JDK для компіляції за умовчанням, навіть якщо Eclipse було сконфігуровано для використання іншої версії JDK.

2.2 Розробка алгоритму конвертації UML в OWL

2.2.1 Дослідження можливостей Apache Jena

Перед початком роботи з бібліотекою Apache Jena необхідно було провести детальний аналіз її можливостей. Була проведена робота з пошуку документації для Apache Jena. Багато часу було виділено на дослідження прикладного програмного інтерфейсу (API). Також не менш важливим фактором виявився пошук існуючих прикладів коду [18] та розуміння основних підходів для роботи з ним.

Детальне дослідження та вивчення можливостей Jena проводилося шляхом спроб та помилок.

Першими результатами роботи з бібліотекою Jena було створення об’єкту OntoModel та можливість експорту цієї моделі у формат RDF, а згодом і проведення конвертації у формат OWL.

2.2.2 Дослідження об'єктів даних (фігур), а також їх особливостей в ArgoUML

На цьому етапі було проведено детальне дослідження UML діаграм. За їх побудову та відображення відповідає класс ArgoDiagram. За всю необхідну інформацію для елементів відповідають класи ModelElement та Figures.

UML елементи (класи): UmlClass, UmlAssociation, Generalization, Abstraction, Stereotype, та інші.

Отримавши ці класи було визначено як вони будуть використані для конвертації у формат OWL згідно встановленим правилам у PSI [2].

UmlClass містить в собі об’єкт типу Attribute. За допомогою якого можна отримати перелік атрибутів цього класу, їх тип, ім’я та іншу додаткову інформацію. Тобто отримавши об’єкти діаграми у вигляді UmlClass цей

елемент буде конвертований у OWL як <owl:Class>, а його власні атрибути як <owl:DatatypeProperty>

Враховуючи той факт що зв’язки типу асоціація, агрегація та композиція не відрізняються в OWL, вся необхідна інформація може бути визначена з використанням належної множини PSI [19], конвертація UmlAssociation буде проводитися використовуючи слідуючі елементи в залежності від типу:

<owl:Class>, <rdfs:subClassOf>, <owl:Restriction>, <owl:maxCardinality>, <owl:minCardinality>, <owl:allValuesFrom>, <owl:inverseOf>

Всі ці елементи будуть конвертуватися у відповідні OWL сутності в залежності від параметрів Multiplicity, що містяться у UmlAssociation.

2.2.3 Написання алгоритму перетворення ArgoUML діаграм (фігур) у класи і залежності OWL

Після проведення аналізу UML діаграми, наступним етапом буде її конвертація у формат мови онтологій. Опираючись на документ з правилами конвертації PSI [2] вона буде проводитися у наступному порядку:

а) отримання ModelElement з поточної діаграми UML;

б) конвертація класів <owl:Classrdf:ID=“ClassName”>..</owl:Class>;

в) конвертація атрибутив;

<owl:DatatypeProperty rdf:ID=“ClassName-description”>

<rdfs:range rdf:resource=“http://www.w3.org/2001/XMLSchema#int”/>

<rdfs:domain rdf:resource=“#ClassName"/>

<rdf:type rdf:resource="http://www.w3.org/2002/owlFunctionalProperty"/>

г) конвертація залежностей відбувається на основі поточних параметрів відношення multiplicity. Наприклад відношення zero-to-many буде виглядати наступним чином у форматі .owl. додаток Д;

д) на наступному етапі буде конвертована генералізація. У цьому випадку залежний об’єкт буде виглядати як субклас іншого об’єкту;

<owl:Class rdf:about="http://www.w3.org/2002/07/owl#ClassName1">
<rdfs:subClassOf>
<owl:Class rdf:about="http://www.w3.org/2002/07/owl#ClassName2"/>
</rdfs:subClassOf>
</owl:Class>

е) далі за допомогою бібліотеки Apache Jena будуть створені об’єкти OntClass, OntModel, OntProperty, Resource та інші;

ж) на останньому етапі конвертовані дані будуть записані у файл з розширенням .owl.

2.2.4 Складнощі при розробці алгоритму

Досить працеємким процесом виявився пошук відповідних елементів поточної діаграми в ArgoUML. Багато спроб були марними. Дані які були отримані на початку не відповідали вимогам, а також вони не містили всю необхідну інформацію для проведення конвертації у OWL. Але після детального дослідження коду було знайдено класи які відповідають за надання повної моделі UML з поточної діаграми (діаграми яка відкрита у проекті ArgoUML).

Протягом довгого часу не вдавалося домогтися коректної конвертації у формат OWL згідно правилам PSI [2]. Було вирішено зупинитися на форматі RDF, але нарешті вдалося отримати прийнятний результат за допомогою спільноти відкритого коду Apache Jena. Але надане рішення виявилося не стандартним і може перестати коректно працювати у майбутньому.

Багато часу вирішувалася проблема з формуванням атрибутів. А саме складнощі при додаванні типу.

<rdf:type rdf:resource="http://www.w3.org/2002/07
owl#FunctionalProperty"/>

Результати які були отримані не були коректними, замість додавання типу, було отримано його перевизначення, що виглядало наступним чином:

<owl:FunctionalProperty
rdf:about=“http://www.w3.org/2002/07/owl#Role-description">
<rdf:type rdf:
resource="http://www.w3.org/2002/07owl#DatatypeProperty"/>
</owl:FunctionalProperty>

Проблемою виявилися недоліки у написаному на той момент коді. Працюючи над залежностями досить складно було конвертувати різні типи відношень таких як one-to-one, one-to-many, zero-to-many, zero-to-one. Вирішення цієї задачі поділялося на два етапи: перший – це пошук визначення різних відношень між класами поточної діаграми ArgoUML, другий – пошук шляхів конвертації цих відношень згідно документу PSI [2].

Проблемою яка не вирішена й досі є спосіб представлення класів у форматі OWL. Застосовуючи вбудований конвертор Apache Jena, замість властивості ID з’являється властивість about що не відповідає правилам конвертації документу PSI [2].

Таблиця 2.1 – Порівняння очікуваного та отриманого результату конвертації класу у OWL формат

Очікуваний результат	Поточний результат
<owl:Class rdf:ID="Role"> ... </owl:Class>	<owl:Class rdf:about=“Role"/>

2.3 Програмна реалізація розширення ArgoUML

2.3.1 Впровадження бібліотеки Apache Jena

Для впровадження бібліотеки Apache Jena був використаний менеджер залежностей Maven. Впровадження бібліотеки Jena було необхідним тільки для модулю argouml-app та argouml-core-model-mdr. Наступний код був доданий у pom.xml цього модулю.

```
<dependency>
        <groupId>org.apache.jena</groupId>
        <artifactId>apache-jena-libs</artifactId>
        <version>3.2.0</version>
        <type>pom</type>
</dependency>
```

2.3.2 Встановлення відповідних залежностей і запуск проекту на jdk 8

Оскільки для управління залежностями (бібліотеки та фреймворки) був використаний інструмент Maven, це означало що пошук та встановлення необхідних розширень буде виконуватися у автоматичному режимі.

Наприклад, для встановлення бібліотек за допомогою командного інтерфейсу (консолі), достатньо виконати mvn install. Є і інший варіант, це використання графічного інтерфейсу Eclipse та додаткового плагіну Maven.

Наступним розглянемо запуск проекту на JDK 1.8 у режимі сумісності з JDK 1.6. Для цього необхідно встановити відповідний JDK у меню Build Path –> Edit Library яке показано на малюнку нижче.

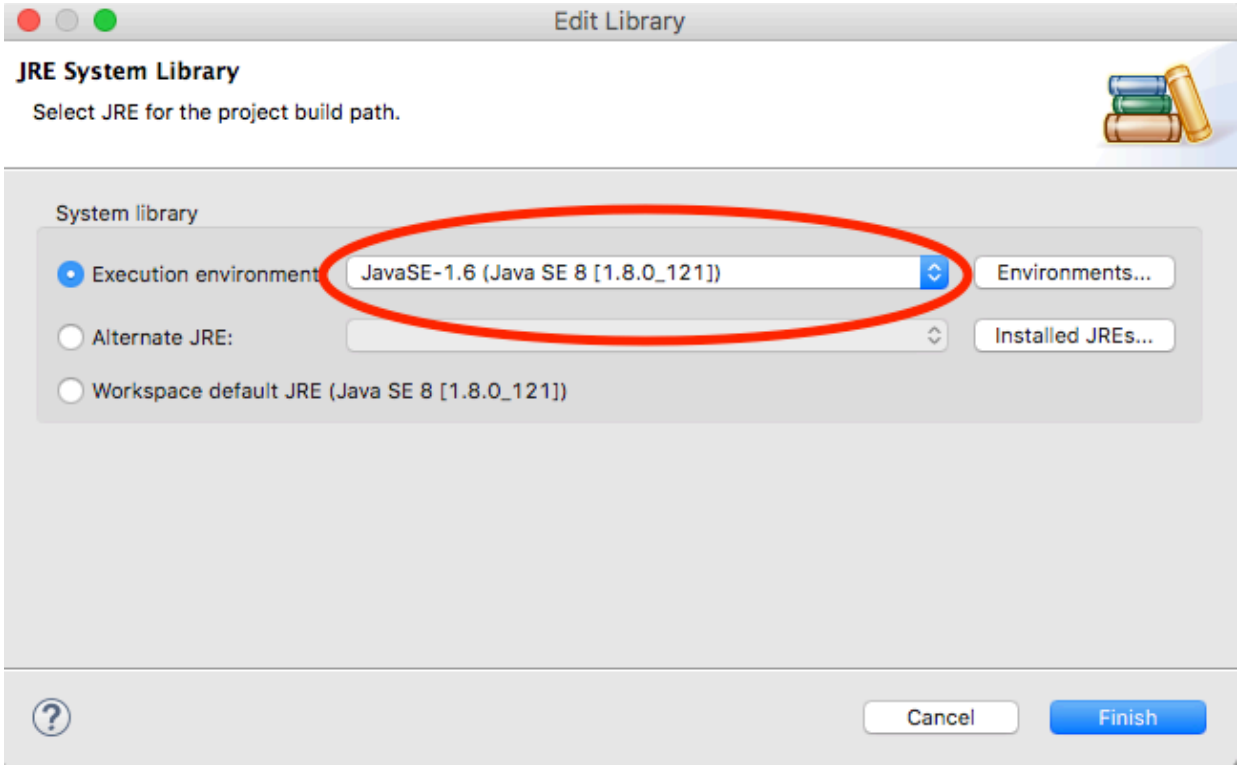


Рисунок 2.4 – Меню Edit Library у Eclipse

А також необхідно змінити середовище виконання Java (JRE), залишити тільки встановлений JDK 1.8 і видалити системний JDK 1.6. Наступні дії потрібно зробити тільки якщо розробка ведеться на MacOS. В операційних системах Windows та Linux, декілька JRE можуть співіснувати у Eclipse не викликаючи при цьому помилок.

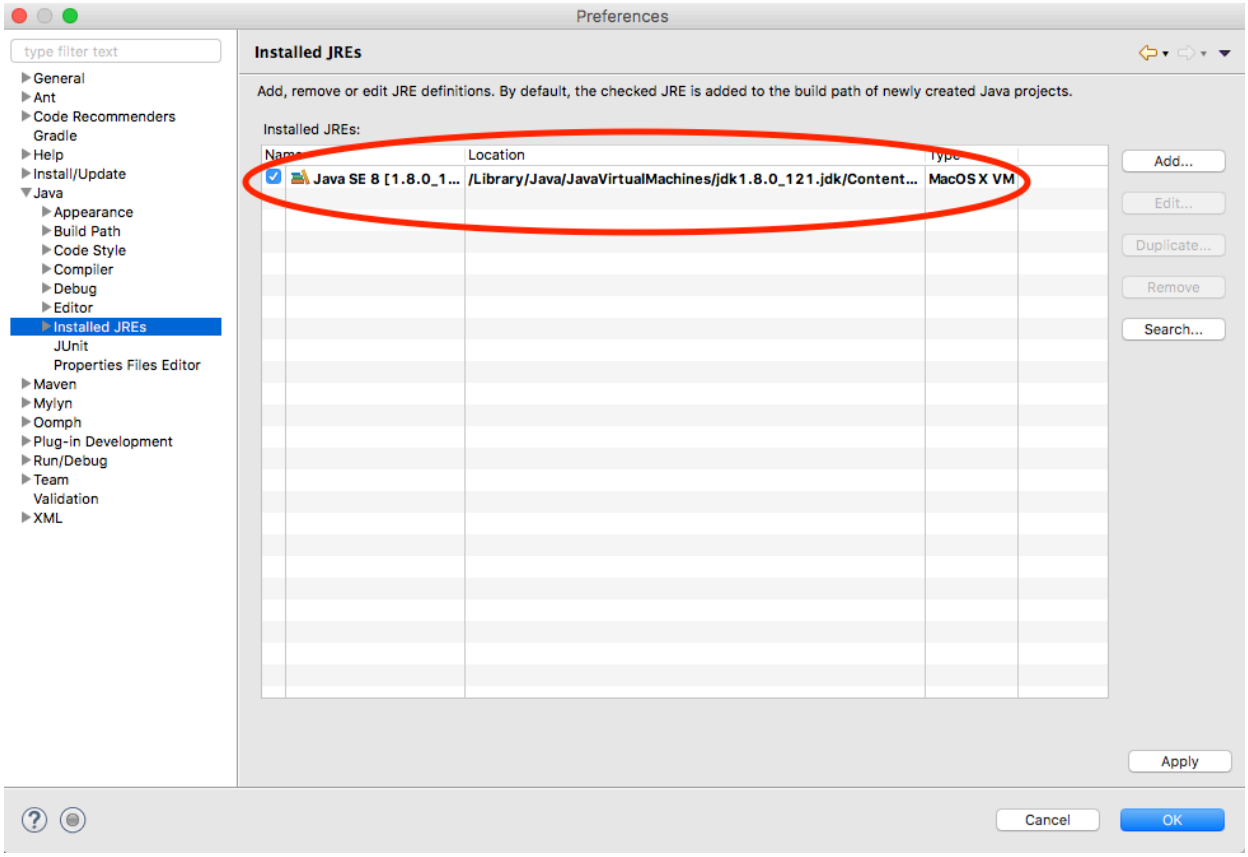


Рисунок 2.5 – Меню Installed JREs у Eclipse

2.3.3 Розширення користувацького інтерфейсу

Для розширення інтерфейсу, а саме для створення нового пункту меню Export OWL… був створений класс ActionExportOWL. Код якого наведений нижче.

```
public void actionPerformed(ActionEvent e) {
    PersistenceManager pm = PersistenceManager.getInstance();
    JFileChooser chooser = new JFileChooser();
    chooser.setDialogTitle(Translator.localize(
    "action.export-project-as-owl"));
    chooser.setFileView(ProjectFileView.getInstance());
```

```
        chooser.setApproveButtonText(
        Translator.localize( "filechooser.export"));
        chooser.setAcceptAllFileFilterUsed(true);
        pm.setXmiFileChooserFilter(chooser);
        …
}
```

Наступним було проведене розширення класу GenericArgoMenuBar із додаванням створеного класу ActionExportOWL для створення необхідного пункту меню.

```
ShortcutMgr.assignAccelerator(file.add(new ActionExportOWL()),
ShortcutMgr.ACTION_EXPORT_OWL);
```

Також були проведені зміни тексту для пункту меню, які знаходяться у файлі action.properties.

```
action.export-project-as-owl = Export OWL...
```

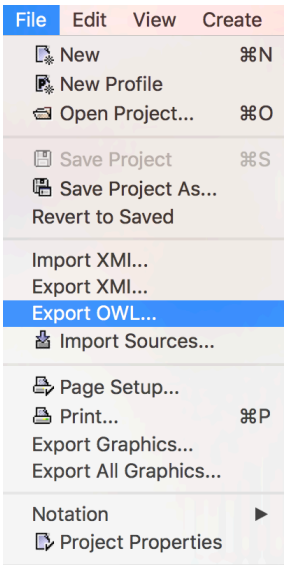


Рисунок 2.6 – Меню File у ArgoUML після розширення інтерфейсу

2.3.4 Створення класів для конвертації UML в OWL із застосуванням розробленого алгоритму

Для конвертації елементів UML у формат OWL був створений класс OwlWriterMDRImp який реалізує інтерфейс OwlWriter. Саме тут застосовуючи ModelManagementHelper я отримав елементи поточної UML діаграми, такі як класи і залежності.

Приклад коду для отримання елементів поточної UML діаграми можно побачити у додатку Е.

Далі було проведено детальний аналіз та отримання інформації з кожного елементу діаграми UML, для подальшого перетворення цих елементів на сутності в OWL за допомогою використання API з бібліотеки Apache Jena.

Для початку конвертації у формат OWL, необхідно було створити об'єкт типу OntModel

```
OntModel ontModel=
ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);
```

Параметр OWL_MEM – вказує на формат у який буде виконуватися конвертація. Далі необхідно буде створити класи OntClass або їх властивості типу DatatypeProperty, та приєднати їх до створеної раніше онто моделі.

На етапі конвертації класів та їх атрибутів, необхідно ітегрувати список атрибутів для отримання детальної інформації по кожному з них (додаток Ж).

При побудові зв’язку типу генералізація, необхідно було отримати Parent та Child об’єкти які містяться у поточному зв’язку, та побудувати відповідне відношення OWL сутностей (класів), спираючись на дані з цих об’єктів. Приклад коду можна побачити нижче.

```
private void convertUmlGeneralization (OntModel ontModel, Object item) {
        Generalization generalizationObj = (Generalization) item;
        OntClass classParent = ontModel.createClass(OWL.NS +
                generalizationObj.getParent().getName());
        OntClass classChild = ontModel.createClass(OWL.NS +
                generalizationObj.getChild().getName());
        classParent.setSubClass(classChild);
}
```

При конвертації залежностей необхідно було створити Restriction відношення одного класу до іншого, а також вказати мінімальні та максимальні показники. У випадку відношення one-to-many або zero-to-many необхідно встановити параметр allValuesFromRestriction. Приклад коду конвертації відношення one-to-many:

```
private void createOneToMany(OntModel ontoModel, OntClass ontoClass,
        OntProperty prop, Resource resource) {
        OntClass allValuesFromRestriction =
                ontoModel.createAllValuesFromRestriction(
                        null, prop, resource);
        ontoClass.addSuperClass(allValuesFromRestriction);
        OntClass minCardinalityRestriction =
                ontoModel.createMinCardinalityRestriction(null, prop, 1);
        ontoClass.addSuperClass(minCardinalityRestriction);
}
```

Останній етап – це зберігання створеної онтології у файл з розширенням .owl

```
modelImpl.getRepository().beginTrans(false);
ontModel.write(oStream, “RDF/XML-ABBREV");
```

2.3.5 Впровадження створених класів в інфраструктуру ArgoUML

Для того щоб додати створені класи у поточну структуру ArgoUML, треба максимально дотримуватися існуючих архітектурних рішень. Тобто класи та інтерфейси створювалися аналогічно тим, які містяться у проекті ArgoUML, що спростило процес розширення та впровадження нових функцій.

Треба також зазначити, що редактор ArgoUML хоч і складається з кількох модулів (суб-проектів), але його архітектура залишається монолітною, вона не містить механізму розширень та плагінів, що в свою чергу ускладнює додавання нових функцій, а також вимагає попереднього знання, внутрішніх механізмів взаємодії між класами.

У разі подальшого розвитку даного проекту, система плагінів буде завданням номер один, у списку необхідних удосконалень.

2.4 Висновки

У цьому розділі було описано процес дослідження проекту ArgoUML та пошук варіантів його розширення. Абстрактну схему розширеня можно побачити на рисунку 2.3. Наступними були розглянуті переваги та недоліки досліджених підходів, а також методи впровадження нового функціоналу з урахуванням поточної архітектури проекту.

У завершальній частині цього розділу, було розглянуто програмний механізм реалізації розширення ArgoUML, інтеграцію бібліотеки Jena та алгоритм конвертації UML діаграми в онтологію на мові OWL. Результатом якого вдалося досягти у цьому розділі можна вважати, розширений редактор

ArgoUML, який набуває змоги конвертувати UML діаграми у OWL онтології. Також ArgoUML допомагає аналізувати та редагувати знання з будь якої предметної області у вигляді UML діаграм, що є першим кроком на шляху до отримання коректних OWL онтологій згідно з правилами вибраної предметної області. Даний результат, буде використаний в наступному розділі у якості бази для проведення експериментальної перевірки відповідності OWL онтологій до вимог.

3 ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА РОБОТИ ARGOUML ПІСЛЯ РОЗШИРЕННЯ ФУНКЦІОНАЛЬНОСТІ

У цьому розділі будуть розглянуті етапи експериментальної перевірки проекту. Будуть наведені основні підготовчі частини для проведення експерименту та показані результати його роботи. Також буде розглянутий план подальшого покращення та надання нового функціоналу поточному проекту, приблизні шляхи для його впровадження, а також нові ідеї які можуть бути втілені в життя у наступних версіях програмного продукту.

3.1 Підготовка до експерименту

3.1.1 Отримання вихідних даних парсера тексту у форматі XMI

Перш за все нам потрібно підготувати тексти різного типу для парсингу з наступною конвертацією у формат XMI [1]. Таким чином будуть отримані дані для аналізу та створення онтологій OWL.

Необхідно знайти декілька текстів для конвертації в XMI, умовою буде те що ці тексти повинні бути семантично насичені, а також не мати помилок як орфографічних так і синтаксичних. У разі якщо ж тексти будуть мати велику кількість помилок, програма парсер не зможе надати прийнятну UML діаграму для аналізу та створення онтології OWL. Наприклад, буде взятий куплет із вірша [20]

The woods are lovely, dark and deep,
But I have promises to keep,
And miles to go before I sleep.

А також декілька семантично насичених текстів з ресурсу SOT-Clock [21]:

а) A Clock is a TemporalInstrument to generate the instances of a TemporalMeasure.

б) A Clock, as a measurement instrument, may return a single value (a TimeStamp corresponding to a single TimeUnit) or several values (the parts of a TimeStamp corresponding to different TimeUnits).

XMI код після конвертації куплету вірша можна побачити у додатку А.

Переконавшись у тому, що на даному етапі було отримано прийнятну діаграму UML, переходимо до наступного етапу, імпорту діаграми з відображенням її у ArgoUML.

3.1.2 Імпорт XMI файлу

Для подальшого аналізу та редагування діаграми, необхідно імпортувати XMI файл через меню імпорту. Наступним потрібно додати діаграму класів у робочу область ArgoUML, це можна зробити як показано на рисунку 3.1 натиснувши правою кнопкою миші на імпортованій діаграмі та вибравши наступний пункт у меню “Add All classes in Namespace”.

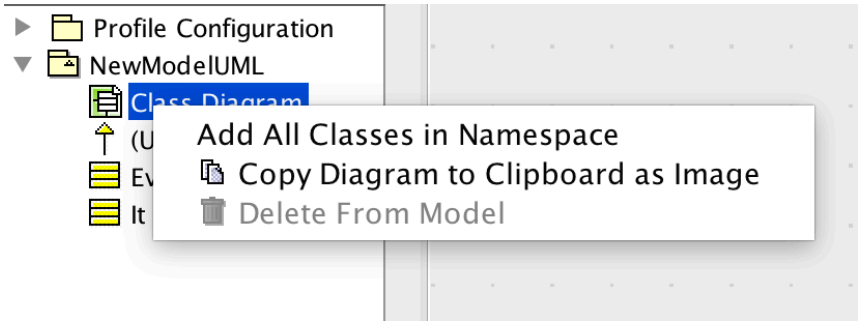


Рисунок 3.1 – Додавання UML діаграми до робочої області ArgoUML

У разі некоректного розташування елементів UML діаграми, після її відображення у робочій області ArgoUML, можна спробувати скористатися алгоритмом авторозміщення з дипломної роботи [22]. На рисунку 3.2 показано як це можна зробити за допомогою меню Arrange –> Smart Layout.

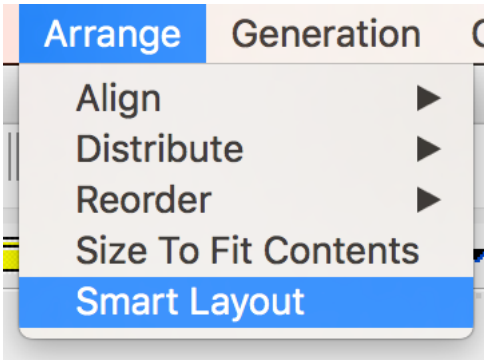


Рисунок 3.2 – Авторозміщення елементів діаграми

3.1.3 Аналіз та перевірка адекватності імпортованих даних

Отримавши прийнятну діаграму UML, з попередньо підготовленого тексту природною мовою, та провівши імпорт в ArgoUML, можна переходити до наступного етапу. Цим етапом буде аналіз на прийнятність імпортованих даних. Цей тип аналізу буде виконуватися користувачем. Також користувач може виконувати необхідні правки у UML діаграмі за допомогою інструментів редагування ArgoUML.

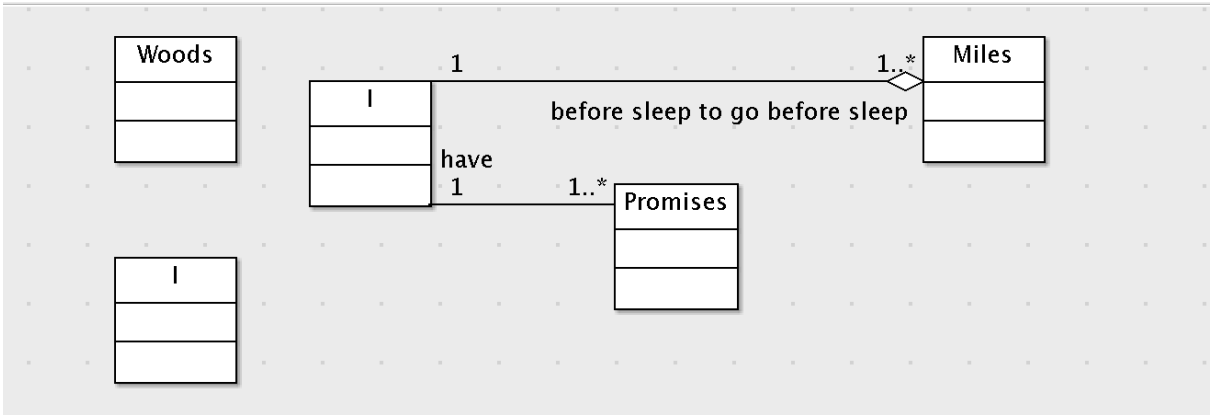


Рисунок 3.3 - UML діаграма після імпорту в ArgoUML

UML діаграма на рисунку 3.3 виглядає не зовсім коректною, наприклад знаючи текст, та відношення між дієвими особами, можна було б видалити дубльований клас I. Але відповідні корективи будуть проведені при подальшому аналізі. На даному етапі цікавить точність співпадіння UML діаграми при імпортуванні з XMI, тому порівнявши та проаналізувавши класи, їх атрибути та зв'язки між ними, врахувавши типи, відношення та інші параметри, можна сказати що UML діаграма була коректно імпортована.

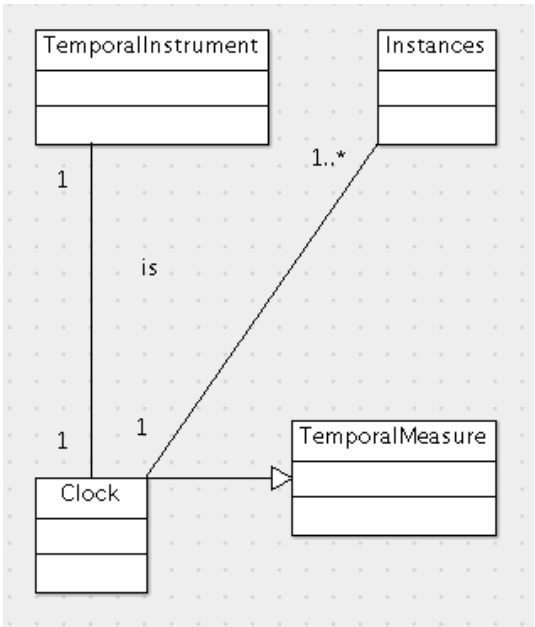


Рисунок 3.4 – UML діаграма для першого тексту з SOT-Clock [21]

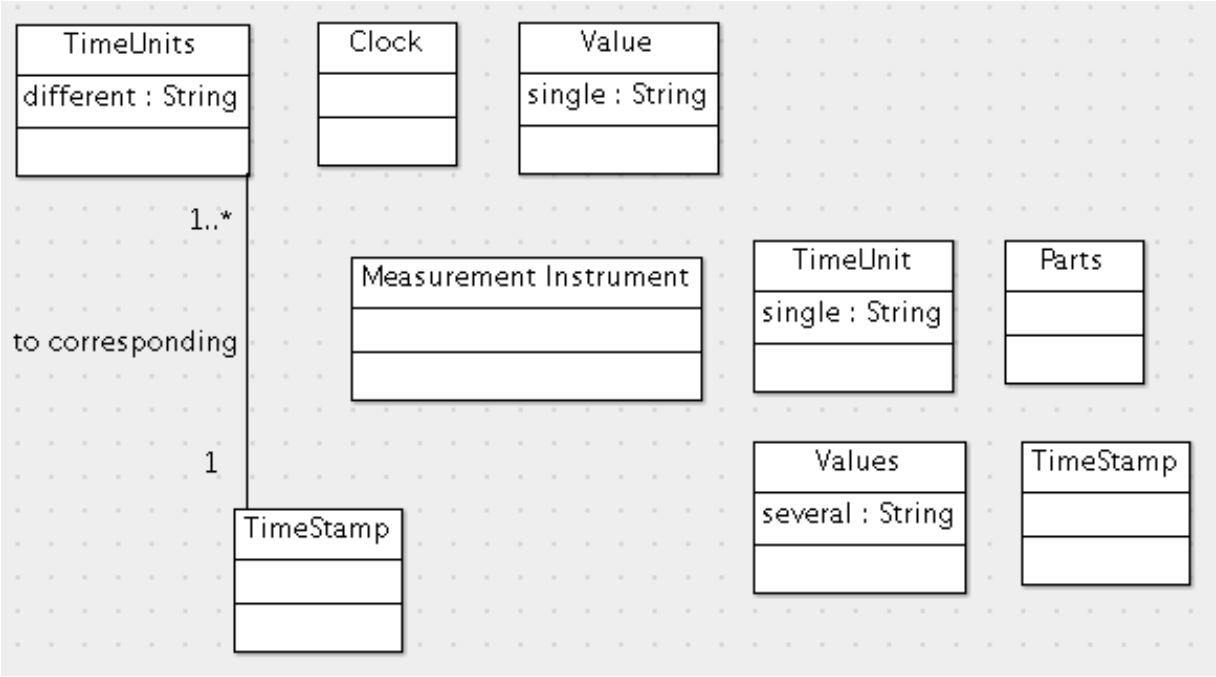


Рисунок 3.5 – UML діаграма для другого тексту з SOT-Clock [21]

UML діаграми на рисунках 3.4 - 3.5 які були отримані з семантично насичених текстів [21] виглядають також коректними, але у діаграми на рисунку 3.5 відсутня переважна більшість зв’язків між класами. Також на діаграмі (рисунок 3.4) присутній клас Instances який може бути видалений при подальшому редагуванні для набуття створюваною онтологією сенсу у відповідності до вимог предметної області.

3.2 Етапи проведення експерименту

3.2.1 Аналіз і редагування UML діаграми класів засобами ArgoUML, відповідно до вимог предметної області

Наступним етапом є виконання аналізу отриманої діаграми, шляхом перевірки та порівняння з встановленими правилами. Якщо ж діаграма не відповідає цим правилам, буде проведене її редагування.

Давайте проаналізуємо діаграму на рисунку 3.3 та проведемо необхідні правки, якщо такі потрібні.

Після імпортування діаграми, можна бачити що клас I дублюється. Тож логічно буде видалити зайві копії, та якщо буде потрібно, провести необхідні корективи у залежностях.

Порівняння атрибутів: Треба перевірити перелік атрибутів та обов’язково звернути увагу на їх тип. При більш детальному аналізі та порівнянні діагрми із рядками віршу, можна помітити відсутність потенційних атрибутів “lovely, dark and deep” у класі Woods. Порівняння зв’язків та залежностей:

а) перевірити коректність зв’язків, видалити зайві або додати необхідні;

б) проаналізувати тип, та змінити у разі необхідності, на відповідний;

в) провести перевірку коректності відношень. Наприклад, якщо один з класів вказує на однину I, а інший на множину Promises то відношення повинно бути one-to-many;

г) проаналізувати імена зв’язків. У прикладі на рисунку 3.3 ім’я зв’язку містить два раза фразу “before sleep”.

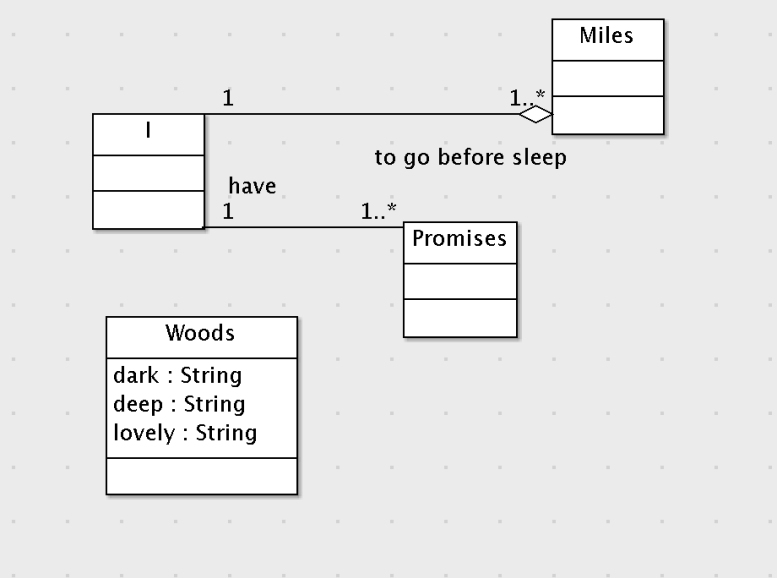


Рисунок 3.6 – Результат аналізу та редагування

Результат редагування діаграми можна побачити на рисунку 3.6 Ця діаграма виглядає вже більш прийнятною для створення онтології OWL.

Також розглянемо діаграми на рисунках 3.4 - 3.5 Після редагування діаграма на рисунку 3.5 буде виглядати наступним чином рисунок 3.7.

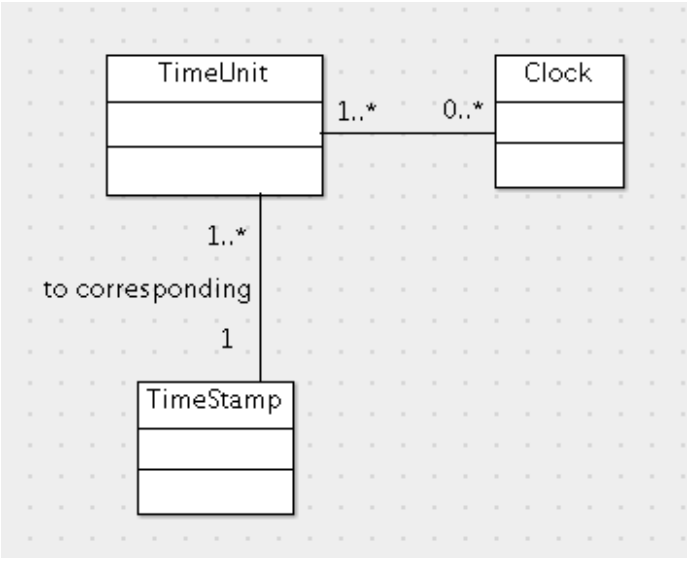


Рисунок 3.7 – Результат редагування другого тексту з SOT-Clock [21]

Наприкінці розглянемо діаграму на рисунку 3.8 яка є результатом редагування діаграми на рисунку 3.4.

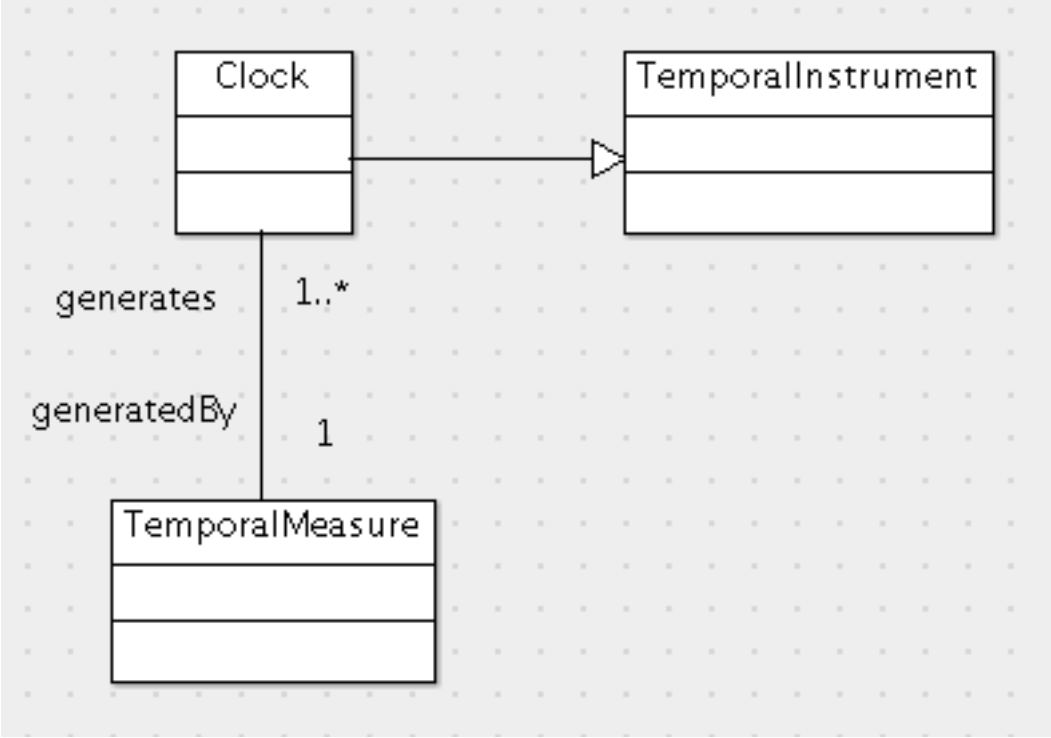


Рисунок 3.8 – Результат аналізу та редагування першого тексту з SOT-Clock [21]

Базуючись на отриманих результатах представлених на рисунках 3.7 - 3.8 можна стверджувати що був проведений графічний аналіз UML діаграм з наступними редагуванням (рисунок 3.9) у відповідності до вимог предметної області, структурна діаграма для SOT-Clock [21] зображена у додатку В. Вона була використана в якості правил у відповідності до яких проводилося редагування діаграм на рисунках 3.7 - 3.8.

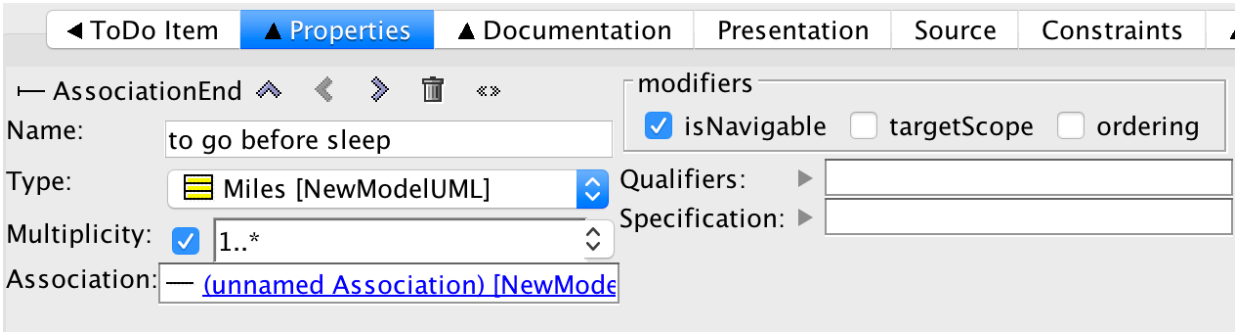


Рисунок 3.9 – Вікно редагування властивостей в ArgoUML

3.2.2 Експорт відредагованої діаграми у формат OWL

Коли аналіз та редагування діаграми проведені, наступним кроком буде її експортування у формат OWL, та збереження у файлі з розширенням .owl. Графічний інтерфейс для експортування у OWL зображений на рисунку 3.10. Результат конвертування вірша [20] в OWL можна побачити у додатку Б.

Наступним кроком буде перевірка експортованих даних (онтологій), яка буде проведена у наступному пункті.

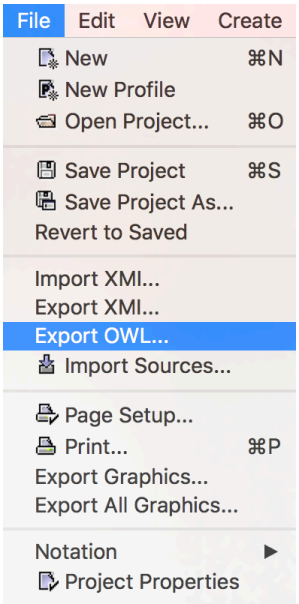


Рисунок 3.10 – Експорт UML діаграми у формат OWL

3.3 Аналіз і оцінка

3.3.1 Використовуємо Protégé для візуального представлення та перевірки даних

Однією з найважливіших частин цього розділу є проведення аналізу конвертованих UML діаграм. А саме проведення порівняльного аналізу згідно документу PSI [2] та визначення відхилень у результатах конвертації між отриманою онтологією та очікуваною.

Для проведення повного аналізу та перевірки конвертованого OWL файлу онтології, найкращим шляхом буде його візуалізація. У підрозділі 1.3 було надано опис інструменту Protégé, а також у додатоку И були надані порівняльні характеристики Protégé та інших інструментів для роботи з онтологіями.

1. Порівняємо відображення класів у Protégé

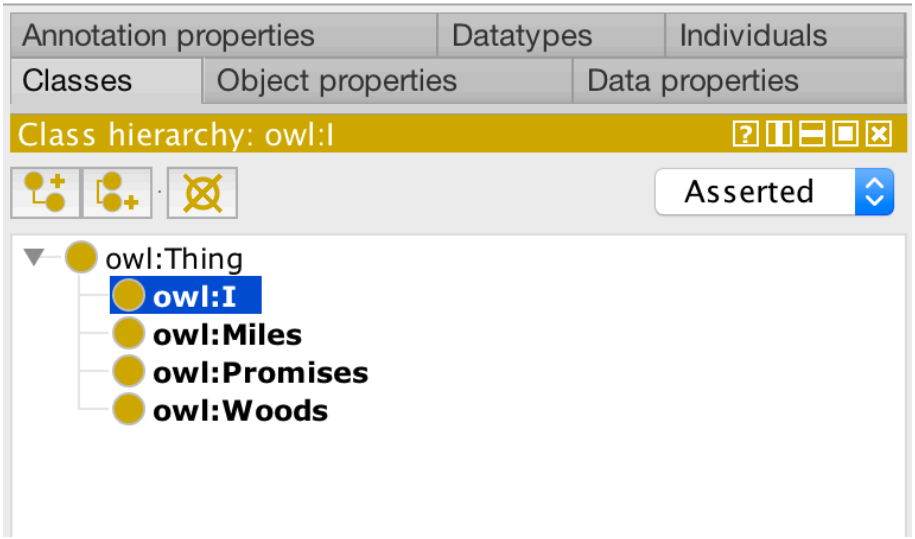


Рисунок 3.11 – Відображення класів у Protégé

Якщо порівняти UML діаграму на рисунку 3.6 як результат аналізу та редагування імпортованих даних, і список отриманих класів у Protégé

(рисунок 3.11) то можна побачити що список та діаграма мають по 4 класи: I, Miles, Promises, Woods. Це означає що класи у OWL співпадають з класами в UML діаграмі.

2. Порівняємо відображення зв'язків. Розглянемо зв'язок між класами I і Miles, як можна побачити на рисунку 3.12 ці класи мають зв'язок типу агрегація.

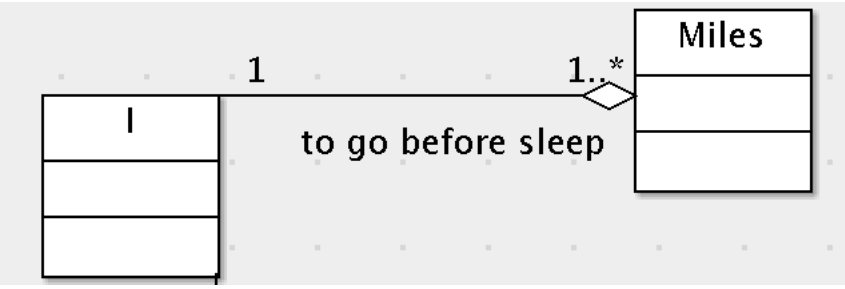


Рисунок 3.12 – Відображення зв'язків у UML

Клас I відноситься до класу Miles як one-to-one. Згідно правилам [19] відношення між цими класами було конвертовано правильно, це можна побачити на рисунку 3.13.

Рисунок 3.13 – Відображення зв'язку класу I до Miles у Protégé

Відношення класу Miles до класу I є зв'язком many-to-one. Цей зв'язок зображений у Protégé відповідно до правил PSI [23], а також вказано позначення AssociationEnd.

Рисунок 3.14 – Відображення зв'язку класу Miles до класу I у Protégé

3. На цьому етапі порівняємо атрибути. Розглянемо атрибут dark, можна зауважити, що конвертація пройшла успішно, згідно з вимогами [23]. Потрібно нагадати, що тут є одна особливість для конвертації атрибутів – це додавання суфіксу -description для атрибутів у OWL.

Рисунок 3.15 – Відображення властивостей у ArgoUML

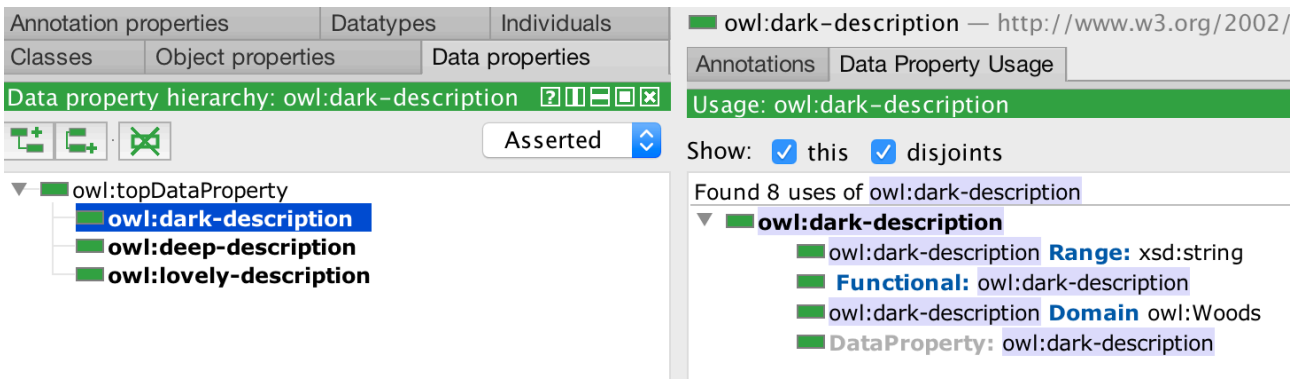


Рисунок 3.16 – Відображення властивостей у Protégé

4. Перевірка конвертації зв’язку типу генералізація. На діаграмі UML (рисунок 3.17) відображено клас Cat який є нащадком класу Animal. Тому, згідно правил конвертації PSI [2], можна зазначити що ієрархія класів у Protégé відображена коректно. Тобто клас Cat є підкласом (SubClassOf) класу Animal.

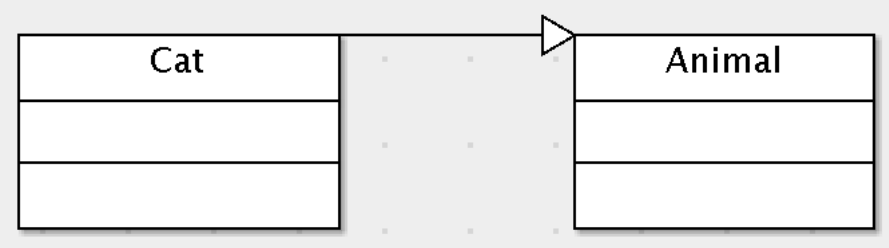


Рисунок 3.17 – Відображення генералізації у UML

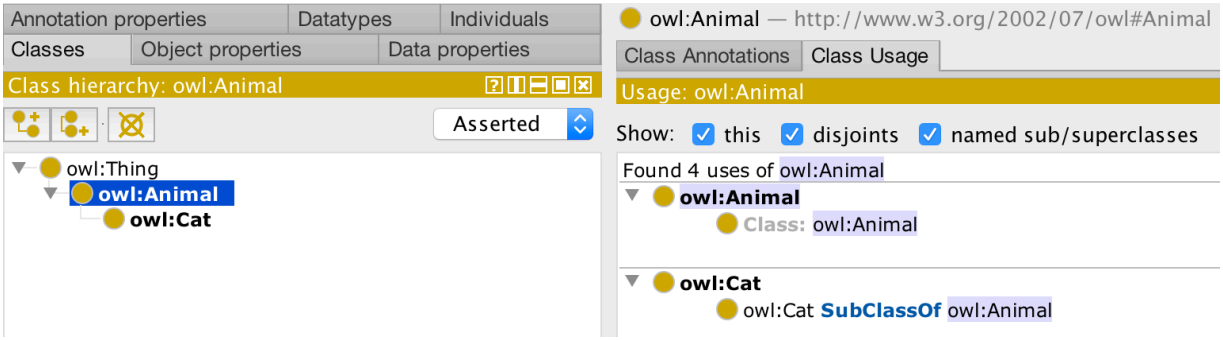


Рисунок 3.18 – Відображення генералізації у Protégé

Після детального аналізу, шляхом візуального порівняння елементів, можна зробити висновок, що конвертація діаграми UML у формат OWL відбулася коректно.

3.3.2 Перевірка відповідності OWL даних встановленим правилам конвертації

У додатку З представлені порівняльні характеристики між очікуваним результатом за правилами PSI [2] та отриманим результатом конвертації UML діаграми у формат OWL.

3.3.3 Аналіз отриманих онтологій у відповідності до вимог предметної області

Розглянемо процес графічного аналізу онтологій на відповідність до вимог предметної області. Відображення онтології у Protégé можна бачити на рисунку 3.19, OWL код знаходиться у додатку Г.

Перше що потрібно зробити, це отримати вимоги (правила) з необхідної предметної області. Вимоги [21] можна побачити у додатку В. Вони представлені у вигляді UML діаграми класів.

Наступним буде аналіз цих вимог та отримання UML діаграми з тексу природною мовою, який належить до вибраної предметної області, за допомогою програми парсера [1].

Далі потрібно імпортувати отриману UML діаграму у редактор ArgoUML та додати елементи цієї діаграми у робочу область для аналізу та редагування. Після успішного редагування діаграми можна переходити до наступного кроку.

Наступним кроком є, експортування (рисунок 3.10) відредагованої UML діаграми у формат OWL, таким чином створюється онтологія, яку буде проаналізовано у відповідності до вимог [21] додаток В.

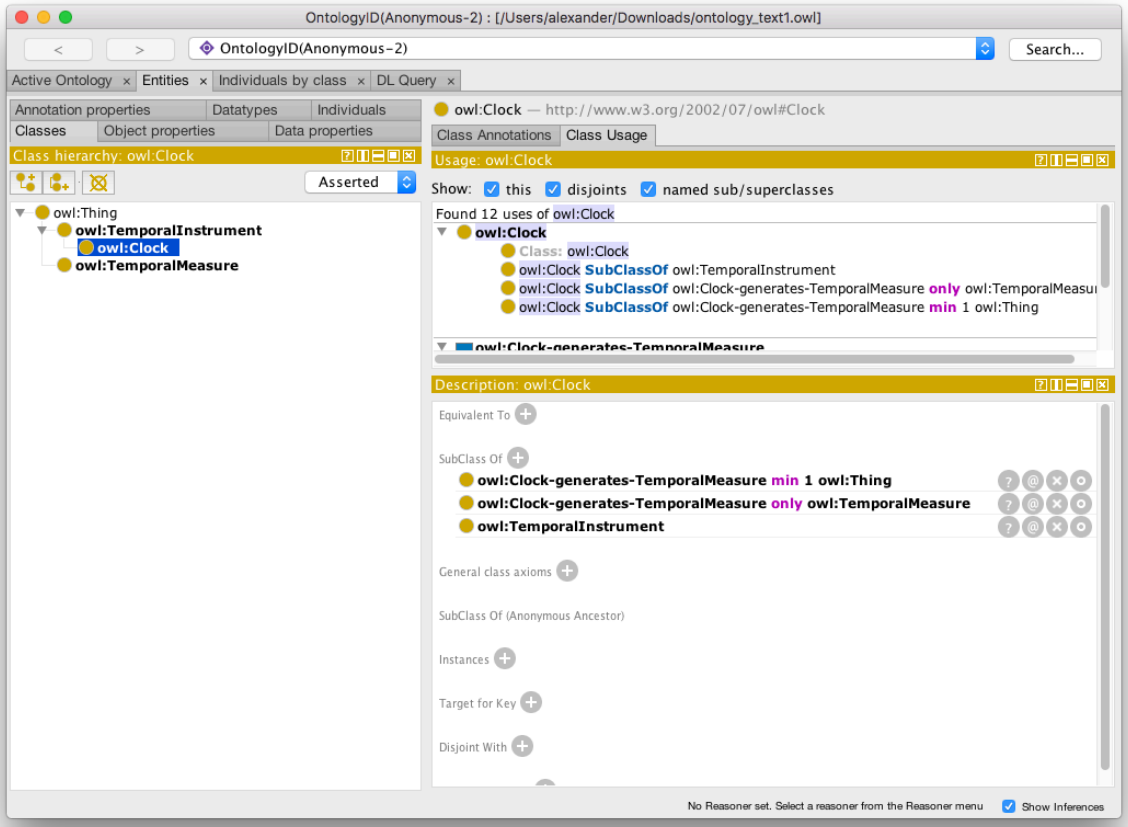


Рисунок 3.19 – Відображення класів онтології у Protégé, отриманих при експортуванні діаграми (рисунок 3.8)

UML діаграма була конвертована у OWL онтологію за правилами, які описані у документі PSI [2] та [21]. Так як UML та OWL є тотожними мовами для репрезентації даних, то при такому типі конвертації втрати інформації будуть мінімальними. Це в свою чергу надає можливість точно проаналізувати відповідність онтології до вимог предметної області.

Наступним буде конвертовано UML діаграму у OWL онтологію з додатку В, таким чином ми можемо отримати правила для порівняння. UML діаграму з додатку В відтворену у ArgoUML можна побачити на рисунку 3.20.

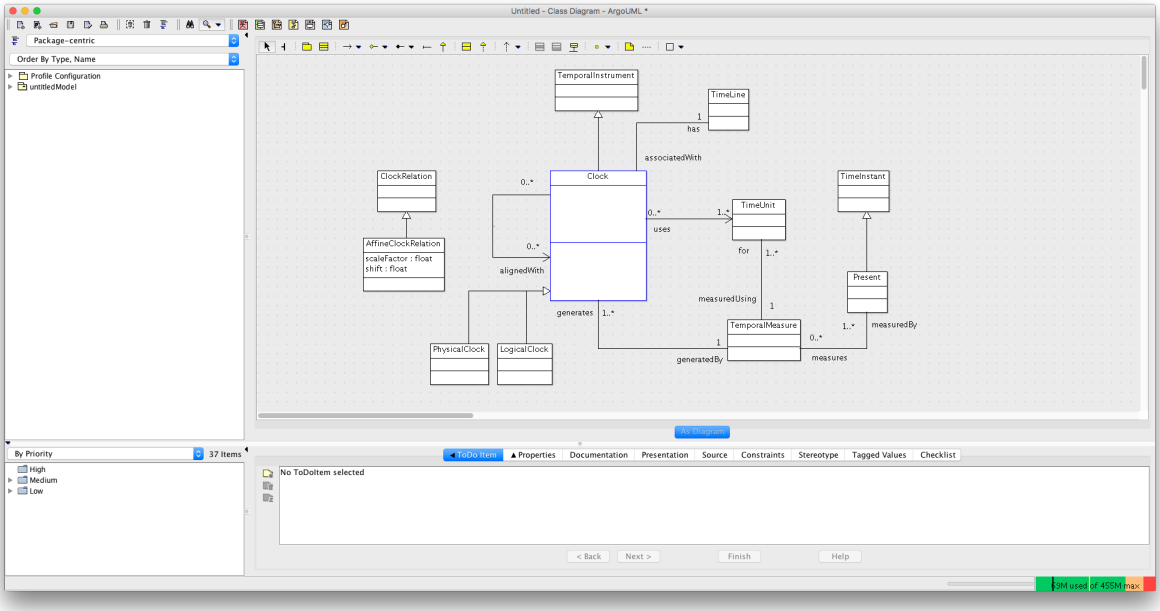


Рисунок 3.20 – UML діаграма з додатку В. відтворена у ArgoUML

Після конвертування правил у OWL формат (конвертовано за допомогою ArgoUML), можемо почати графічний аналіз отриманої онтології з тексту природною мовою [1] на відповідність вимогам. Для цього процесу було обрано OWLDiff плагін встановлений як розширення для інструменту Protégé.

Як можна бачити на рисунку 3.21 отримана онтологія з діаграми (рисунок 3.8) значно звужує область знань та видаляє значну кількість елементів (класів), але це є позитивним результатом для даного проекту, тому що, графічний аналіз був проведений у відповідності до вимог [21]. Що в свою чергу означає результативне проведення експерименту.

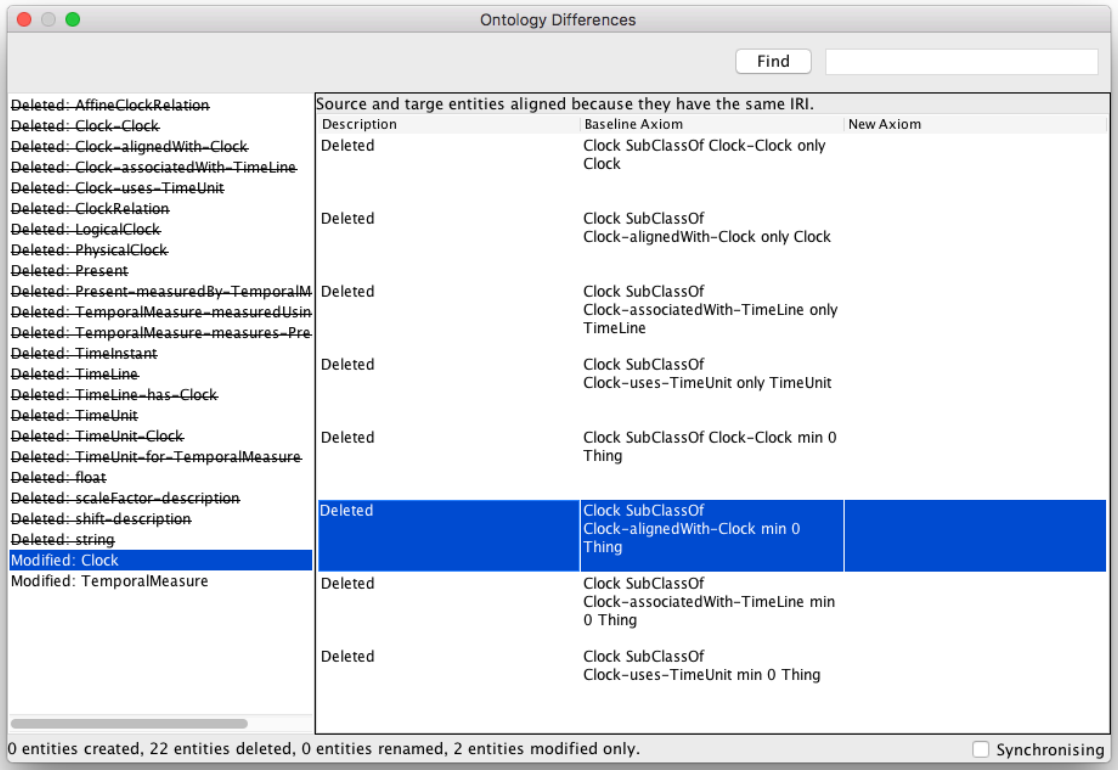


Рисунок 3.21 – Відображення відповідності онтології вимогам у OWLDiff

3.3.4 Визначення шляхів і методів поліпшення роботи розширення в ArgoUML

На данний момент конвертація елементів UML у OWL може виконуватися швидко, тільки при невеликих об’ємах даних. Враховуючи використання лінійних циклів, приблизна швидкість роботи алгоритму конвертації складає $O(n^2)$. Тому наступним кроком планується оптимізувати та покращити цей алгоритм. Ефективним рішенням може бути використання багатопотокової обробки інформації. Але такий підхід потребує змін у “ядрі” ArgoUML що є достатньо працемістким завданням.

Потенційним покращенням є те, що перехід з формату OWL до OWL2 надасть змогу отримати доступ до більш широкого набору функцій та оптимізувати конвертацію UML у формат OWL.

З точки зору користувача, не досить зручним є процес імпорту XMI файлу. Користувач, виконує ті ж самі дії кожнен раз коли імпортує XMI. Цей процес можна оптимізувати та створити сценарій простого імпортування, наприклад: користувач імпортує XMI, елементи UML діаграми будуть автоматично додані на робочу область ArgoUML.

Останнім але не менш важливим є покращення конвертації залежностей відношень one-to-one, one-to-many, zero-to-one, many-to-many враховуючи різні типи зв’язків та їх комбінації між елементами UML діаграми.

3.3.5 Визначення подальших розширень і доробок у ArgoUML

Окрім оптимізації та вдосконалення деяких модулів, було б добре додати декілька нових функцій, щоб зробити цей проект максимально корисним. Нижче буде наведений їх перелік.

Перш за все, необхідно впровадити механізм автоматичного графічного аналізу, на етапі конвертації UML діаграми у мову онтології (OWL). Для імплементації буде використано наступний документ [3] Using Contexts in Ontology Structural Change Analysis

Розширення користувацького інтерфейсу з метою полегшення аналізу та редагування UML діаграм. Наприклад, досить незручним є редагування зв’язків а також їх імен. Потенційним покращенням для аналізу могла б стати функція попереднього перегляду діаграми у вигляді OWL.

Можливість конвертувати UML діаграми не тільки у формат OWL, а також у інші формати такі як RDF(S), XML(S), HTML.

Надання змоги користувачу обирати набір правил для конвертації UML у OWL/RDF під час виконання автоматичного аналізу.

Надати можливість виконувати перевірку конвертованого OWL файлу прямо у ArgoUML без використання Protege. Мабуть це найскладніше розширення для ArgoUML з потенційних.

Імплементувати підсистему плагінів та розширень для ArgoUML, що надасть змогу спільноті відкритого коду, додавати нові функції, а користувачам гнучко їх використовувати. Цей тип розширення є найважливішим на мою думку, тому що саме такі розширення є запорукою успіху та довгого життя проекту.

ВИСНОВКИ

У оглядовій частині були надані основні теоретичні відомості про онтології, їх особливості та сфери застосування. Була надана інформація про мови онтологій, а також був наведений їх синтаксис. На наступному етапі були розглянуті UML діаграми, у якості основної складової частини розробки поточного дипломного проекту. У кінці цього розділу був наданий перелік причин та аргументів вибору інструменту ArgoUML як середовища для проведення аналізу, редагування та конвертації UML у формат OWL.

У основній частині описана специфіка реалізації та перелік труднощів на етапі розробки програмного продукту. Проведений аналіз різних підходів щодо вирішення нагальних питань та проблем з редактором ArgoUML.

Досить важливою, та працемісткою частиною, виявилося отримання UML елементів з поточної діаграми у ArgoUML та їх конвертація у мову онтологій. Також важливим етапом був пошук бібліотеки для конвертації UML у OWL, такою бібліотекою стала Apache Jena.

У результаті цих дій, було знайдено підхід, для створення та реалізації алгоритму конвертації UML діаграм у OWL онтології, використовуючи бібліотеку Apache Jena.

У експериментальній частині були надані результати роботи програмного продукту у вигляді OWL онтологій і проаналізовані вхідні дані у вигляді UML діаграм. Також була розглянута послідовність дій користувача для отримання OWL файлу з наступним аналізом у Protégé. На заключному етапі цього розділу був проведений детальний аналіз та перевірка відповідності отриманих онтологій OWL з встановленими правилами.

У результаті проведеної роботи можна стверджувати, що мета була успішно досягнута. Програмний продукт допомагає вирішувати проблему аналізу онтологій відповідно з вимогами вибраної предметної області. Але як і кожний програмний продукт він має багато можливостей для подальшого

розширення, наприклад, система плагінів, яка б могла значно полегшити наступні удосконалення. Також дуже привабливим є надання можливості динамічної зміни правил конвертації у процесі роботи програми. Всі ці вдосконалення та розширення будуть впроваджені у майбутньому з залученням спільноти відкритого коду, що може надати друге життя для ArgoUML, а також розширити його аудиторію.

ПЕРЕЛІК ПОСИЛАНЬ

1. Моісеєнко С.А. Розробка програмного забезпечення для парсингу текстів і генерації UML моделей [Електронний ресурс] Режим доступу: https://github.com/alexnodejs/onto-diploma
2. Ermolayev V., Keberle N., Jentasch E., Sohnius R.: Performance Simulation Initiative . Upper-Level Ontology v.2.3 Reference Specification, 2009, VCAD EMEA Cadence Design Systems GmbH
3. Ermolayev V., Copylov A., Keberle N., Jentasch E., Matzke W.-E.: Using Contexts in Ontology Structural Change Analysis, Zaporozhye National University, Cadence Design Systems GmbH
4. What is an Ontology? [Електронний ресурс] Режим доступу: http://www-ksl.stanford.edu/kst/what-is-an-ontology.html
5. Ontology [Електронний ресурс] Режим доступу: https://en.wikipedia.org/wiki/Ontology_(information_science)
6. OWL, an ontology language [Електронний ресурс] Режим доступу: http://ontogenesis.knowledgeblog.org/55
7. Web Ontology Language OWL [Електронний ресурс] Режим доступу: https://www.obitko.com/tutorials/ontologies-semantic-web/web-ontology-language-owl.html
8. Emhimed Alatrish, Comparison Some of Ontology Editors, 2013 [Електронний ресурс] Режим доступу: http://www.ef.uns.ac.rs/mis/archive-pdf/2013%20-%20No2/MIS2013-2-4.pdf
9. Ontology Development 101: A Guide to Creating Your First Ontology [Електронний ресурс] Режим доступу: http://protege.stanford.edu/publications/ontology_development/ontology101-noy-mcguinness.html
10. Knublauch H., Fergerson R.W., Noy N.F., Musen M.A. (2004) The Protégé OWL Plugin: An Open Development Environment for Semantic Web

Applications. In: McIlraith S.A., Plexousakis D., van Harmelen F. (eds) The Semantic Web – ISWC 2004. ISWC 2004. Lecture Notes in Computer Science, vol 3298. Springer, Berlin, Heidelberg

11. The Protégé OWL Plugin, Holger Knublauch, July 07 2004 [Електронний ресурс] Режим доступу: https://protege.stanford.edu/conference/2004/slides/Knublauch_2004-07-07-OWL-Plugin.pdf
12. A configurable translation-based cross-lingual ontology mapping system to adjust mapping outcomes [Електронний ресурс] Режим доступу: http://www.websemanticsjournal.org/index.php/ps/article/view/294
13. Ontology alignment [Електронний ресурс] Режим доступу: https://en.wikipedia.org/wiki/Ontology_alignment#Ontology_alignment_methods
14. UML [Електронний ресурс] Режим доступу: https://ru.wikipedia.org/wiki/UML
15. List of Unified Modeling Language tools [Електронний ресурс] Режим доступу: https://en.wikipedia.org/wiki/List_of_Unified_Modeling_Language_tools
16. Alejandro Ramirez, Philippe Vanpeperstraete, Andreas Rueckert, Kunle Odutola, Jeremy Bennett, Linus Tolke, and Michiel van der Wulp, ArgoUML User Manual : A tutorial and reference description, 2000 - 2004
17. Apache Jena [Електронний ресурс] Режим доступу: https://jena.apache.org/
18. Java Code Examples [Електронний ресурс] Режим доступу: http://www.programcreek.com/java-api-examples/index.php?api=com.hp.hpl.jena.ontology.OntClass
19. Falkovych K., Sabou M., and Stuckenschmidt H.: UML for the Semantic Web: Transformation-Based Approaches. 2003, [Електронний ресурс] Режим доступу: http://citeseer.ist.psu.edu/falkovych03uml.html

20. Robert Frost, Stopping by Woods on a Snowy Evening [Електронний ресурс] Режим доступу: https://www.poetryfoundation.org/poems-and-poets/poems/detail/42891
21. SOT-Clock [Електронний ресурс] Режим доступу: http://km.aifb.kit.edu/sites/qs-sdm/wiki/SOT-Clock
22. Печерський В.Н. Розробка програмного забезпечення для оптимізації розташування елементів UML діаграм [Електронний ресурс] Режим доступу: https://github.com/alexnodejs/onto-argouml
23. Schreiber G. OWL Restrictions [Електронний ресурс] Режим доступу: http://www.cs.vu.nl/~guus/public/owl-restrictions/
24. Ermolayev V., Keberle N., Harth A.: Part I: Refining Temporal Representations using OntoElect. In: Extended Semantic Web Conference, 2016 Heraklion, Crete, May 29, 2016
25. Ermolayev V.: Gravitation and Fitness in Ontology Dynamics. In: KIT, AIFB: Kolloquium Angewandte Informatik Jan. 26, 2016, Karlsruhe, Germany
26. Vadim Ermolayev V.: Toward a Syndicated Ontology of Time for the Semantic. In: KIT, AIFB February, 2016, Karlsruhe, Germany
27. Thomas R. Gruber, A Translation Approach to Portable Ontology Specifications [Електронний ресурс] Режим доступу: https://pdfs.semanticscholar.org/e790/2a46a83aa52ff8e2a36578a25f720fa648a2.pdf
28. Web Ontology Language [Електронний ресурс] Режим доступу: https://en.wikipedia.org/wiki/Web_Ontology_Language
29. Ontology Alignment Evaluation Initiative [Електронний ресурс] Режим доступу: http://oaei.ontologymatching.org/
30. UML Modeling Language [Електронний ресурс] Режим доступу: http://www.uml.org/
31. Xmi2owl [Електронний ресурс] Режим доступу: https://github.com/alexnodejs/xmi2owl

Додаток А

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<XMI xmlns:UML="org.omg.xmi.namespace.UML" xmi.version="1.2">
    <XMI.header>
        <XMI.metamodel xmi.name="UML" xmi.version="1.4"/>
    </XMI.header>
    <XMI.content>
        <UML:Model xmi.id="UML_model_1" name="NewModelUML"
isSpecification="false" isRoot="false" isLeaf="false"
isAbstract="false">
            <UML:Namespace.ownedElement>
                <UML:Class visibility="public" isActive="false"
xmi.id="Woods_ClassID0" name="Woods" isSpecification="false"
isRoot="false" isLeaf="false" isAbstract="false"/>
                <UML:Class visibility="public" isActive="false"
xmi.id="I_ClassID2" name="I" isSpecification="false" isRoot="false"
isLeaf="false" isAbstract="false"/>
                <UML:Class visibility="public" isActive="false"
xmi.id="Promises_ClassID3" name="Promises" isSpecification="false"
isRoot="false" isLeaf="false" isAbstract="false"/>
                <UML:Class visibility="public" isActive="false"
xmi.id="Miles_ClassID4" name="Miles" isSpecification="false"
isRoot="false" isLeaf="false" isAbstract="false"/>
                <UML:Class visibility="public" isActive="false"
xmi.id="I_ClassID5" name="I" isSpecification="false" isRoot="false"
isLeaf="false" isAbstract="false"/>
                <UML:Association xmi.id="have_AssociationID6"
name="have" isSpecification="false" isRoot="false" isLeaf="false"
isAbstract="false">
                    <UML:Association.connection>
                        <UML:AssociationEnd
xmi.id="_assocEndFor_I_ClassID26" visibility="public"
isNavigable="true" ordering="unordered" aggregation="none"
targetScope="instance" changeability="changeable"
isSpecification="false" isRoot="false" isLeaf="false"
isAbstract="false">
                            <UML:AssociationEnd.multiplicity>
<UML:Multiplicity xmi.id="multiplicity__assocEndFor_I_ClassID26">
    <UML:Multiplicity.range>
        <UML:MultiplicityRange xmi.id="range__assocEndFor_I_ClassID26"
lower="1" upper="1"/>
    </UML:Multiplicity.range>
</UML:Multiplicity>
                            </UML:AssociationEnd.multiplicity>
                            <UML:AssociationEnd.participant>
<UML:Class xmi.idref="I_ClassID2"/>
                            </UML:AssociationEnd.participant>
                        </UML:AssociationEnd>
                        <UML:AssociationEnd
xmi.id="_assocEndFor_Promises_ClassID36" visibility="public"
isNavigable="true" ordering="unordered" aggregation="none"
targetScope="instance" changeability="changeable"
```

```
isSpecification="false" isRoot="false" isLeaf="false"
isAbstract="false">
                            <UML:AssociationEnd.multiplicity>
<UML:Multiplicity
xmi.id="multiplicity__assocEndFor_Promises_ClassID36">
    <UML:Multiplicity.range>
        <UML:MultiplicityRange
xmi.id="range__assocEndFor_Promises_ClassID36" lower="1" upper="-1"/>
    </UML:Multiplicity.range>
</UML:Multiplicity>
                            </UML:AssociationEnd.multiplicity>
                            <UML:AssociationEnd.participant>
<UML:Class xmi.idref="Promises_ClassID3"/>
                            </UML:AssociationEnd.participant>
                        </UML:AssociationEnd>
                    </UML:Association.connection>
                </UML:Association>
                <UML:Association xmi.id="_AssociationID7" name=""
isSpecification="false" isRoot="false" isLeaf="false"
isAbstract="false">
                    <UML:Association.connection>
                        <UML:AssociationEnd
xmi.id="_assocEndFor_Miles_ClassID47" visibility="public"
isNavigable="true" ordering="unordered" aggregation="aggregate"
targetScope="instance" changeability="changeable" name="before sleep
to go before sleep" isSpecification="false" isRoot="false"
isLeaf="false" isAbstract="false">
                            <UML:AssociationEnd.multiplicity>
<UML:Multiplicity xmi.id="multiplicity__assocEndFor_Miles_ClassID47">
    <UML:Multiplicity.range>
        <UML:MultiplicityRange
xmi.id="range__assocEndFor_Miles_ClassID47" lower="1" upper="-1"/>
    </UML:Multiplicity.range>
</UML:Multiplicity>
                            </UML:AssociationEnd.multiplicity>
                            <UML:AssociationEnd.participant>
<UML:Class xmi.idref="Miles_ClassID4"/>
                            </UML:AssociationEnd.participant>
                        </UML:AssociationEnd>
                        <UML:AssociationEnd
xmi.id="_assocEndFor_I_ClassID27" visibility="public"
isNavigable="true" ordering="unordered" aggregation="none"
targetScope="instance" changeability="changeable" name=""
isSpecification="false" isRoot="false" isLeaf="false"
isAbstract="false">
                            <UML:AssociationEnd.multiplicity>
<UML:Multiplicity xmi.id="multiplicity__assocEndFor_I_ClassID27">
    <UML:Multiplicity.range>
        <UML:MultiplicityRange xmi.id="range__assocEndFor_I_ClassID27"
lower="1" upper="1"/>
    </UML:Multiplicity.range>
</UML:Multiplicity>
                            </UML:AssociationEnd.multiplicity>
                            <UML:AssociationEnd.participant>
<UML:Class xmi.idref="I_ClassID2"/>
                            </UML:AssociationEnd.participant>
                        </UML:AssociationEnd>
                    </UML:Association.connection>
```

```
                </UML:Association>
                <UML:Association xmi.id="have_AssociationID8"
name="have" isSpecification="false" isRoot="false" isLeaf="false"
isAbstract="false">
                    <UML:Association.connection>
                        <UML:AssociationEnd
xmi.id="_assocEndFor_I_ClassID28" visibility="public"
isNavigable="true" ordering="unordered" aggregation="none"
targetScope="instance" changeability="changeable"
isSpecification="false" isRoot="false" isLeaf="false"
isAbstract="false">
                            <UML:AssociationEnd.multiplicity>
<UML:Multiplicity xmi.id="multiplicity__assocEndFor_I_ClassID28">
    <UML:Multiplicity.range>
        <UML:MultiplicityRange xmi.id="range__assocEndFor_I_ClassID28"
lower="1" upper="1"/>
    </UML:Multiplicity.range>
</UML:Multiplicity>
                            </UML:AssociationEnd.multiplicity>
                            <UML:AssociationEnd.participant>
<UML:Class xmi.idref="I_ClassID2"/>
                            </UML:AssociationEnd.participant>
                        </UML:AssociationEnd>
                        <UML:AssociationEnd
xmi.id="_assocEndFor_Promises_ClassID38" visibility="public"
isNavigable="true" ordering="unordered" aggregation="none"
targetScope="instance" changeability="changeable"
isSpecification="false" isRoot="false" isLeaf="false"
isAbstract="false">
                            <UML:AssociationEnd.multiplicity>
<UML:Multiplicity
xmi.id="multiplicity__assocEndFor_Promises_ClassID38">
    <UML:Multiplicity.range>
        <UML:MultiplicityRange
xmi.id="range__assocEndFor_Promises_ClassID38" lower="1" upper="-1"/>
    </UML:Multiplicity.range>
</UML:Multiplicity>
                            </UML:AssociationEnd.multiplicity>
                            <UML:AssociationEnd.participant>
<UML:Class xmi.idref="Promises_ClassID3"/>
                            </UML:AssociationEnd.participant>
                        </UML:AssociationEnd>
                    </UML:Association.connection>
                </UML:Association>
            </UML:Namespace.ownedElement>
        </UML:Model>
    </XMI.content>
</XMI>
```

Додаток Б

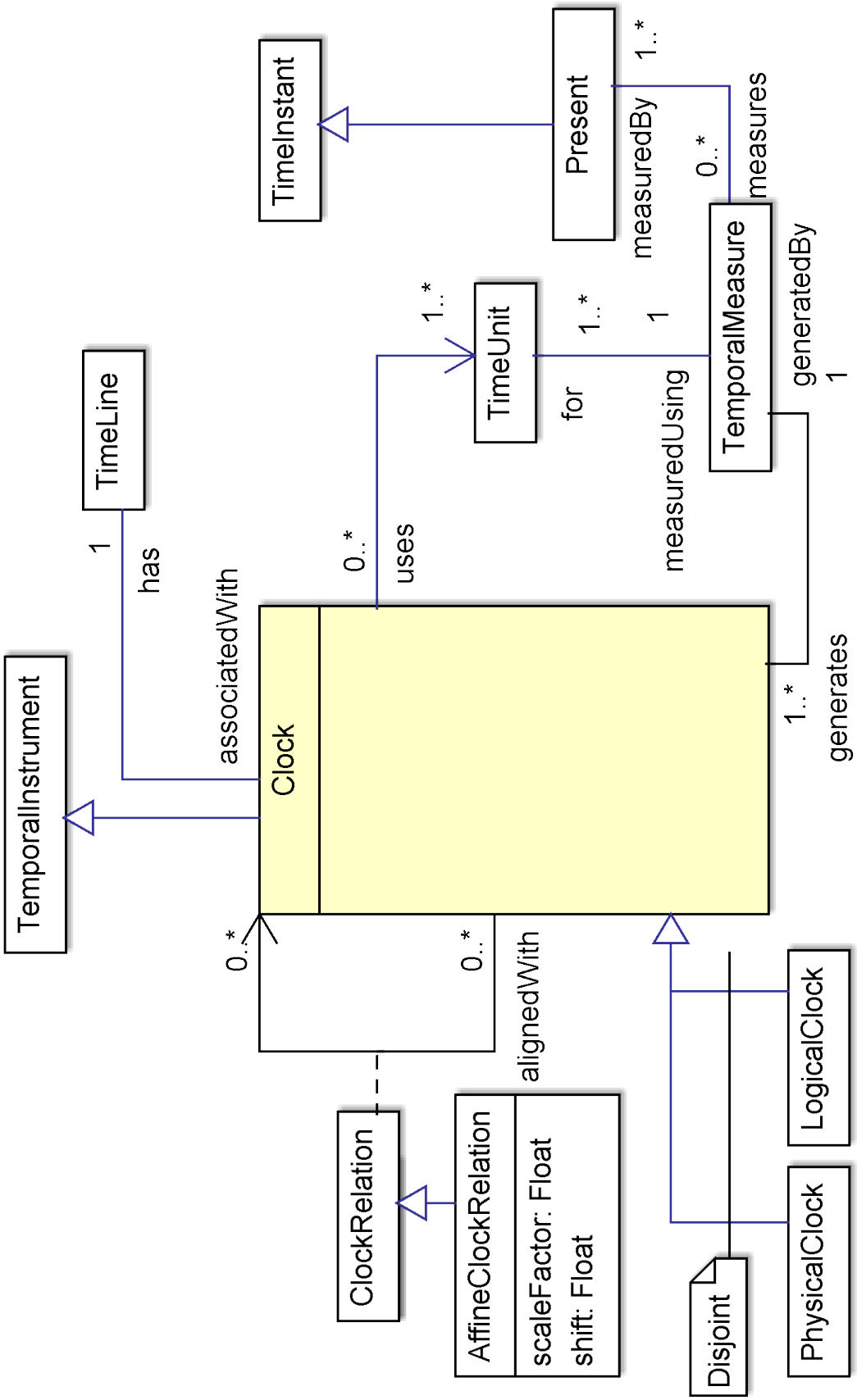
```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
  <owl:Class rdf:about="http://www.w3.org/2002/07/owl#Miles">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:minCardinality>
        <owl:onProperty>
          <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#Miles-to go before sleep-I"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom>
          <owl:Class rdf:about="http://www.w3.org/2002/07/owl#I"/>
        </owl:allValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#Miles-to go before sleep-I"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="http://www.w3.org/2002/07/owl#Woods"/>
  <owl:Class rdf:about="http://www.w3.org/2002/07/owl#Promises">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:minCardinality>
        <owl:onProperty>
          <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#Promises-I"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom>
          <owl:Class rdf:about="http://www.w3.org/2002/07/owl#I"/>
        </owl:allValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#Promises-I"/>
        </owl:onProperty>
      </owl:Restriction>
```

```
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:minCardinality>
        <owl:onProperty>
          <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#Promises-I"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom>
          <owl:Class rdf:about="http://www.w3.org/2002/07/owl#I"/>
        </owl:allValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#Promises-I"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:about="http://www.w3.org/2002/07/owl#I">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:maxCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
        <owl:onProperty>
          <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#I-Promises"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:minCardinality>
        <owl:onProperty>
          <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#I-Promises"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:maxCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
        <owl:onProperty>
          <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#I--Miles"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
```

```
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:minCardinality>
        <owl:onProperty>
          <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#I--Miles"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:maxCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
        <owl:onProperty>
          <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#I-Promises"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:minCardinality>
        <owl:onProperty>
          <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#I-Promises"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:ObjectProperty rdf:about="http://www.w3.org/2002/07/owl#I--
Miles">
    <owl:inverseOf>
      <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#Miles-to go before sleep-I"/>
    </owl:inverseOf>
    <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#Miles"/>
    <rdfs:domain rdf:resource="http://www.w3.org/2002/07/owl#I"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="http://www.w3.org/2002/07/owl#I-
Promises">
    <owl:inverseOf>
      <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#Promises-I"/>
    </owl:inverseOf>
    <rdfs:domain rdf:resource="http://www.w3.org/2002/07/owl#I"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2002/07/owl#Promises"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="http://www.w3.org/2002/07/owl#Miles-
to go before sleep-I">
    <owl:inverseOf rdf:resource="http://www.w3.org/2002/07/owl#I--
Miles"/>
    <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#I"/>
```

```
    <rdfs:domain rdf:resource="http://www.w3.org/2002/07/owl#Miles"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#Promises-I">
    <owl:inverseOf rdf:resource="http://www.w3.org/2002/07/owl#I-
Promises"/>
    <rdfs:domain
rdf:resource="http://www.w3.org/2002/07/owl#Promises"/>
    <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#I"/>
  </owl:ObjectProperty>
  <owl:DatatypeProperty rdf:about="http://www.w3.org/2002/07/owl#deep-
description">
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="http://www.w3.org/2002/07/owl#Woods"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty
rdf:about="http://www.w3.org/2002/07/owl#lovely-description">
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="http://www.w3.org/2002/07/owl#Woods"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
  <owl:DatatypeProperty rdf:about="http://www.w3.org/2002/07/owl#dark-
description">
    <rdf:type
rdf:resource="http://www.w3.org/2002/07/owl#FunctionalProperty"/>
    <rdfs:domain rdf:resource="http://www.w3.org/2002/07/owl#Woods"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  </owl:DatatypeProperty>
</rdf:RDF>
```

Додаток В


TimeInstant
Present
1..*
measuredBy
0..*
measures
TimeUnit
1..*
1..*
for
1
measuredUsing
TemporalMeasure
generatedBy
1
TimeLine
1
has
associatedWith
0..*
uses
TemporalInstrument
Clock
1..*
generates
0..*
0..*
alignedWith
ClockRelation
AffineClockRelation
scaleFactor: Float
shift: Float
LogicalClock
PhysicalClock
Disjoint

Додаток Г

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#">
  <owl:Class rdf:about="http://www.w3.org/2002/07/owl#Clock">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:minCardinality>
        <owl:onProperty>
          <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#Clock-generates-
TemporalMeasure"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom>
          <owl:Class
rdf:about="http://www.w3.org/2002/07/owl#TemporalMeasure"/>
        </owl:allValuesFrom>
        <owl:onProperty>
          <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#Clock-generates-
TemporalMeasure"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Class
rdf:about="http://www.w3.org/2002/07/owl#TemporalInstrument"/>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class
rdf:about="http://www.w3.org/2002/07/owl#TemporalMeasure">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:maxCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:maxCardinality>
        <owl:onProperty>
          <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#TemporalMeasure-generatedBy-
Clock"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
```

```
        <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:minCardinality>
        <owl:onProperty>
          <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#TemporalMeasure-generatedBy-
Clock"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#TemporalMeasure-generatedBy-
Clock">
    <owl:inverseOf>
      <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#Clock-generates-
TemporalMeasure"/>
    </owl:inverseOf>
    <rdfs:range rdf:resource="http://www.w3.org/2002/07/owl#Clock"/>
    <rdfs:domain
rdf:resource="http://www.w3.org/2002/07/owl#TemporalMeasure"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:about="http://www.w3.org/2002/07/owl#Clock-
generates-TemporalMeasure">
    <owl:inverseOf
rdf:resource="http://www.w3.org/2002/07/owl#TemporalMeasure-
generatedBy-Clock"/>
    <rdfs:range
rdf:resource="http://www.w3.org/2002/07/owl#TemporalMeasure"/>
    <rdfs:domain rdf:resource="http://www.w3.org/2002/07/owl#Clock"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

Додаток Д

```
<owl:Class rdf:about="http://www.w3.org/2002/07/owl#ClassName1">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:minCardinality
rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >0</owl:minCardinality>
        <owl:onProperty>
          <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#ClassName1-ClassName2"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:allValuesFrom
rdf:resource="http://www.w3.org/2002/07/owl#ClassName2"/>
        <owl:onProperty>
          <owl:ObjectProperty
rdf:about="http://www.w3.org/2002/07/owl#ClassName1-ClassName2"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
```

Додаток Е

```
ModelManagementHelper helper =
      modelImpl.getModelManagementHelper();
      Iterator itr = helper.getContents(model).iterator();

OntModel ontModel =
           ModelFactory.createOntologyModel(OntModelSpec.OWL_MEM);

           while (itr.hasNext()) {
                Object item = itr.next();
                if (item instanceof UmlClass) {
                     convertUmlClass(ontModel, item);
                } else if (item instanceof UmlAssociation) {
                     convertUmlAssociation(ontModel, item);
                } else if (item instanceof Generalization) {
                     convertUmlGeneralization(ontModel, item);
                } else if (item instanceof Abstraction) {
                     convertUmlAbstraction(ontModel, item);
                } else if (item instanceof Stereotype) {
                     convertUmlStereotype(ontModel, item);
                }
           }
```

Додаток Ж

```
private void convertUmlClass(OntModel ontModel, Object item) {
        UmlClass classObj = (UmlClass) item;
        OntClass ontoClass = ontModel
           .createClass(OWL.NS + classObj.getName());

        Iterator itrAttr = classObj.getFeature().iterator();
        while (itrAttr.hasNext()) {
                Object itemFeature = itrAttr.next();

                if (itemFeature instanceof Attribute) {
                     Attribute attribute = (Attribute) itemFeature;
                     DatatypeProperty datatypeProperty =
                          ontModel.createDatatypeProperty(
                          OWL.NS + attribute.getName() + "-
                               description");
                     datatypeProperty.setDomain(ontoClass);
                     datatypeProperty.setRange(
                               getAttributeType(attribute
                                    .getType().getName()));

                     datatypeProperty.setRDFType(
                          OWL.FunctionalProperty);

     datatypeProperty.addRDFType(OWL.DatatypeProperty);
                }
        }
 }
```

Додаток З

Перелік умовних позначень:

PSI – Performance Simulation Initiative PSI [2]

DIP – Дипломна робота

\+ – Розбіжності присутні

- – Розбіжностей немає (трансформація пройшла коректно)

OWL приклади конвертації класів		Розбіжності
PSI	`<owl:Class rdf:ID="Role"> ...</owl:Class>`	+
DIP	`<owl:Class rdf:about="http://www.w3.org/2002/07/owl#Miles"> … </owl:Class>`	
Порівняння зв'язків		
PSI	`<owl:Class rdf:about="#Task"> <rdfs:subClassOf> <owl:Restriction> <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int">1</owl:minCardinality> <owl:onProperty> <owl:ObjectProperty rdf:ID="Task-performedBy-Actor"/> </owl:onProperty> </owl:Restriction> </rdfs:subClassOf> <rdfs:subClassOf> <owl:Restriction> <owl:onProperty> <owl:ObjectProperty rdf:about="#Task-performedBy-Actor"/> </owl:onProperty> <owl:allValuesFrom rdf:resource=" #Actor"/> </owl:Restriction> </rdfs:subClassOf> </owl:Class>`	-

DIP	<owl:Class rdf:about="http://www.w3.org/2002/07/owl#Miles"> <rdfs:subClassOf> <owl:Restriction> <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int" >1</owl:minCardinality> <owl:onProperty> <owl:ObjectProperty rdf:about="http://www.w3.org/2002/07/owl#Miles-to go before sleep-I"/> </owl:onProperty> </owl:Restriction> </rdfs:subClassOf> <rdfs:subClassOf> <owl:Restriction> <owl:allValuesFrom> <owl:Class rdf:about="http://www.w3.org/2002/07/owl#I"/> </owl:allValuesFrom> <owl:onProperty> <owl:ObjectProperty rdf:about="http://www.w3.org/2002/07/owl#Miles-to go before sleep-I"/> </owl:onProperty> </owl:Restriction> </rdfs:subClassOf> </owl:Class>	
Порівняння атрибутів		
PSI	<owl:DatatypeProperty rdf:ID="Role-description"> <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/> <rdfs:domain rdf:resource="#Role"/> <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalPro perty"/>	-
DIP	<owl:DatatypeProperty rdf:about="http://www.w3.org/2002/07/owl#dark- description"> <rdf:type rdf:resource="http://www.w3.org/2002/07/owl#FunctionalPro perty"/> <rdfs:domain rdf:resource="http://www.w3.org/2002/07/owl#Woods"/> <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/> </owl:DatatypeProperty>	
Порівняння генералізації		
PSI	<owl:Class rdf:ID="Effect"> <rdfs:subClassOf> <owl:Class rdf:ID="Event"/> </rdfs:subClassOf>	-

DIP	<owl:Class rdf:about="http://www.w3.org/2002/07/owl#Animal"/> <owl:Class rdf:about="http://www.w3.org/2002/07/owl#Cat"> <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Animal"/> </owl:Class>	

Додаток И

Особливість	Apollo	OntoEdit	Protégé	Swoop	TopBraid Composer
Розробник	KMI	Ontoprise	SMI (Stanford University)	MND (Maryland University)	TopQuadrant
Доступність	Відкритий код	Ліцензування	Відкритий код	Відкритий код	Ліцензування
Має інші інструменти	Немає	OntoAnnotate, OntoBroker, OntoMat, Semantic and Miner	PROMPT, OKBC, JESS, FaCT and Jena	Немає	Sesame, Jena, Allegro Graph
Можливий експорт у мови	OCML, CLOS	OWL, RDF(s), RIF, SPARQL, F-Logic, Excel	XML(S), RDF(S), OWL, HTML, Java, Clips, F-Logic, SWRL-IQ, Instance Selection, MetaAnalysis,OwlDoc, Queeries (RDF, UML) and backend	RDF(S), OIL, DAML	HTML, UML, XSD, Excel, RDB, Oracle database, RDF, XML, TXT

Особливість	Apollo	OntoEdit	Protégé	Swoop	TopBraid Composer
Можливий імпорт з мов	Apollo Meta Language	XML(S), OWL, Excel, RDF(S), UML2.0, database schemes (Oracle, MS-SQL, DB2, MySQL), Outlook E-mails	XML(S), RDF(S), OWL, HTML, (RDF, UML, XML) backend, text file, RDF file, Excel, BioPortal and DataMaster	OWL, XML, RDF, and text formats	RDFa, WOL, XML(S), RDF(S), XHTML, UML, GRDDL, RDB with D2RQ, Microdata and RDFa Web Data Sites, SPIN, Spreedsheets, Oracle database, text file, RDF file, News Feed, Email and Excel
Архітектура	Standalone	Eclipse client/server	Standalone, Client-server	Web based, Client-server	Standalone, Eclipse plug-in
Розширюваність	Plug-ins	Plug-ins	Plug-ins	Plug-ins	Plug-ins
Резервне копіювання	Немає	Немає	Немає	Немає	Має

Особливість	Apollo	OntoEdit	Protégé	Swoop	TopBraid Composer
Сховище онтологій	Files	DBMS	File and DBMS (JDCB)	As HTML Models	DBMS
KR парадігма моделі знань	Frames (OKBC)	Frames and First Order logic	Frames, First Order logic, SWRL, Metaclasses	OWL	RDF, OWL, SWRL
Axiom мова	Необмежений	Має (F-Logic)	Має (PAL)	OWL-DL	OWL-DL
Мотодологічна підтримка	Немає	Має (Onto-Knowledge)	Немає	Немає	Немає
Вбудований двигун інтерфейсу	Немає	Має (Ontodocker)	Має (PAL)	Немає	WOL, SPARQL, Rule
Інші двигуни для інтерфейсу	Немає	Немає	RACER, FACT, FACT++, F-Logic, Pallet	Pellet and RDF-like	OWLIM, Pellet, Jena Rules, Oracle Rules, SPARQL Rules
Обмеження / Перевірка на консистенцію	Має	Має	Має	Тільки з Reasoner plug-in	Має
Графічна таксономія	Немає	Має	Має	Має	Має
Зум	Немає	Має	Має	Немає	Має

Особливість	Apollo	OntoEdit	Protégé	Swoop	TopBraid Composer
Інструменти для спільної роботи	Немає	Має	Має	Має	Має
Бібліотеки онтологій	Має	Має	Має	Немає	Має