

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ**

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ

Кафедра комп'ютерних наук

КВАЛІФІКАЦІЙНА РОБОТА СПЕЦІАЛІСТА

**на тему: «РОЗРОБКА ПРОГРАМНОГО
ЗАБЕЗПЕЧЕННЯ ДЛЯ ПАРСИНГУ ТЕКСТІВ
І ГЕНЕРАЦІЇ UML МОДЕЛЕЙ»**

Виконав: студент 1 курсу, групи 7.1226-з
спеціальності

122 Комп'ютерні науки та інформаційні технології

(шифр і назва спеціальності)

С.О. Моїсеєнко

(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 парсингу текстів і генерації UML моделей»: 71с., 13 рис., 29 джерел, 5 додатків.

Об'єкт дослідження – визначення правил конвертації тексту природною мовою у UML діаграму.

Мета роботи – розробка правил конвертації тексту природною мовою у формат для обміну даними UML, створення програмного продукту для автоматизованої конвертації тексту з наступним збереженням у форматі XMI.

Метод дослідження – описовий, порівняльний, алгоритмізація, програмування.

Апаратура – персональний комп'ютер, програмне забезпечення.

Результат дипломного проекту – програмний продукт та набір правил для конвертації текстів природною мовою у UML (XMI) формат. Тексти буде конвертовано та збережено у файл з розширенням .xmi. Збережені UML дані будуть представлені у вигляді діаграми класів.

NLP, CORENLP, AI, XML, OWL, UML, XMI, ОНТОЛОГІЯ.

ABSTRACT

Specialist's qualifying paper «The Development of Software for Parsing Texts and Generating their UML Models»: 71 pages, 13 figures, 29 references, 5 supplements.

The object of the study – determining of UML conversion rules for natural language text.

The aim of the study – development of UML conversion rules for the natural language text and software creation for the automatic text conversion with further saving in XMI format.

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

The equipment – personal computer, software.

Result of graduation project – software and sets of rules for the natural language text conversion into the UML (XMI) format. The texts will be converted and saved as .xmi file. Saved UML data will be presented as the class diagram.

NLP, CORENLP, AI, XML, OWL, UML, XMI, ONTOLOGY.

ЗМІСТ

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ	2
РЕФЕРАТ	4
ABSTRACT	5
ЗМІСТ	6
ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ	9
ВСТУП	10
1 ОГЛЯД ОБРОБКИ ПРИРОДНОЇ МОВИ	11
ЗАСОБАМИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	11
1.1 Семантичний аналіз тексту як основа для виконання парсинга	11
1.1.1 Natural Language Processing. Його цілі і призначення.....	11
1.1.2 Найбільш часто досліджувані завдання в NLP	12
1.1.3 Види і підходи семантичного аналізу тексту.....	12
1.1.4 Напрямки використання NLP	14
1.2 Огляд існуючого інструментарію Stanford Core NLP	15
1.2.1 Stanford Core NLP основні можливості	15
1.2.2 Досягнення в цій галузі та можливі етапи розвитку	16
1.2.3 Складність роботи з Natural Language Processing і недоліки існуючих бібліотек	16
1.3 Засоби зберігання і візуального представлення тексту.....	17
1.3.1 XMI (XML Metadata Interchange) стандарт OMG для обміну мета даними.....	17
1.3.2 XML мова розмітки даних. Основні принципи і методи роботи	18
1.3.3 Діаграми UML як один із засобів візуального представлення тексту	19
1.4 Вектор розвитку даного напрямку і його перспективи.....	19
1.5 Висновки.....	20

2 РЕАЛІЗАЦІЯ РОЗПІЗНАВАННЯ ТЕКСТУ І СТВОРЕННЯ	21
UML ДІАГРАМ.....	21
2.1 Специфіка і особливості реалізації	21
2.1.1 Технології, мови програмування, середовище, а також чинники їх вибору	21
2.1.2 Технічні складності при роботі з бібліотекою Core NLP	22
2.1.3 Передбачуваний кінцевий результат та складності його досягнення	23
2.2 Механізм побудови UML діаграм	26
2.2.1 Алгоритм вилучення класів та їх залежностей з результатів парсинга тексту засобами Core NLP	26
2.2.2 Побудова проміжного графа та зазначення відносин між елементами	28
2.2.3 Створення UML графу та пост обробка кожного елемента графа з перетворенням в UML елементи.....	30
2.3 Робота з форматом ХМІ	33
2.3.1 Впровадження та використання javax.xml пакета із стандартного набору розширень Javaх	33
2.3.2 Перетворення класів в ХМІ об'єкти	33
2.4 Програмний механізм реалізації	34
2.4.1 Впровадження бібліотеки CoreNLP	34
2.4.2 Отримання результатів парсинга у вигляді дерева сутностей (NP, VP ...)	36
2.4.3 Побудова проміжного графа на основі отриманого дерева.....	38
2.4.4 Побудова UML діаграми шляхом перетворення елементів базового графа.....	41
2.4.5 Генерація ХМІ файлу з метою зберігання та подальшої передачі даних.....	43
2.5 Переваги і недоліки	46
2.6 Висновки.....	47

3 ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА РЕЗУЛЬТАТІВ РОБОТИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ	48
3.1 Мета експерименту	48
3.2 План і проведення експерименту	48
3.2.1 Вхідні дані	49
3.2.2 Проведення експерименту	50
3.2.3 Отримання вихідних даних.....	51
3.3 Перевірка конвертації текстових сутностей в UML елементи.....	55
3.3.1 Перевірка коректності конвертації іменників (підметів) в класи	55
3.3.2 Перевірка коректності конвертації дієслів (присудків) в залежності.....	56
3.3.3 Перевірка коректності конвертації прикметника в атрибути класу	56
3.3.4 Перевірка коректності конвертації відносин один до багатьох, багато до багатьох та інших	57
3.3.5 Перевірка коректності конвертації типів відносин (association, generalization, aggregation)	57
3.4 Аналіз та оцінка проведеного експерименту	58
3.4.1 Оцінка отриманих результатів	58
3.4.2 Визначення вузьких місць і можливості їх оптимізації.....	58
3.5 Висновки.....	59
ВИСНОВКИ	60
ПЕРЕЛІК ПОСИЛАНЬ.....	62
Додаток А	65
Додаток Б.....	67
Додаток В.....	68
Додаток Г	69
Додаток Д.....	70

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

NLP (ОПМ)	Natural Language Processing, Обробка природної мови
AI	Artificial Intelligence, Штучний інтелект
XML	Extensible Markup Language, Розширювана мова розмітки
XMI	XML Metadata Interchange, XML Обмін метаданими
OWL	Web Ontology Language, Мова опису онтологій
UML	Unified Modeling Language, Уніфікована мова моделювання
IDE	Integrated development environment, Інтегроване середовище розробки
CS (ПЗ)	Computer software, Програмне забезпечення
POS	Part of Speech, Частина мови
JAXB	Java Architecture for XML Binding, Java фреймворк для роботи з XML
POM	Project Object Model, Проект Об'єкт Модель

ВСТУП

Лінгвістична обробка природномовних текстів є однією з центральних проблем інтелектуалізації інформаційних технологій сьогодення. Зусилля багатьох науковців спрямовані на розвиток та отримання результатів у цьому напрямку. Велика кількість країн залучена до фінансової підтримки досліджень та розробки програмного забезпечення в області природної обробки мови. У зв'язку із збільшенням темпу розвитку інтернету та комп'ютерних технологій, ця проблема набуває більшої ваги з кожным днем та постійно потребує нових розробок, інвестицій, та досліджень [1].

Ця кваліфікаційна робота зосереджує у собі роботу з семантично насиченим коротким текстом, з використанням бібліотеки Stanford Core NLP, а також створення UML діаграм на основі цього тексту.

Метою є надання змоги користувачу отримати UML діаграму з тексту природною мовою для подальшого аналізу та редагування.

Предметом дослідження цієї роботи є семантично насичені короткі тексти, а об'єктом – результат роботи бібліотеки Stanford Core NLP, а також результат конвертації цього тексту у UML діаграму.

Інформаційна база (матеріали) Інформаційною базою є документація з природної обробки мови та матеріали з опису структури та формування UML діаграм. Додатковими є матеріали по структурам даних, операціями з ними та їх алгоритмічній обробці.

Практичне значення цієї кваліфікаційної роботи становить, отримання та використання UML діаграм у подальшому аналізі та редагуванні тексту, з метою конвертації у OWL формат (мова онтологій) – кваліфікаційна робота [2].

1 ОГЛЯД ОБРОБКИ ПРИРОДНОЇ МОВИ ЗАСОБАМИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

У цьому розділі буде розглянуто, основні відомості про семантичний аналіз тексту та найбільш часто досліджувані завдання у NLP. Також будуть розглянуті види та підходи аналізу тексту і сучасні напрямки використання NLP.

Далі у данному розділі описується бібліотека обробки природної мови Stanford Core NLP, яка використовується як основний модуль для виконання цієї кваліфікаційної роботи.

Також у даному розділі, я намагаюсь розкрити найбільш суттєві аспекти при зберіганні та відображенні обробленої інформації які безпосередньо задіяні у кваліфікаційній роботі.

1.1 Семантичний аналіз тексту як основа для виконання парсинга

1.1.1 Natural Language Processing. Його цілі і призначення

Natural Language Processing (Обробка природної мови) є одним з центральних напрямків штучного інтелекту у комп'ютерній науці, в основі якого лежить аналіз природномовних текстів [3].

Одним з найважливіших напрямків використання NLP є розробка здатності комп'ютерної програми «розуміти» природну мову використовуючи класифікацію різних лінгвістичних одиниць (текстів, слів, словосполучень, пропозицій), що реалізується, практично у всіх застосуваннях лінгвістичного процесора, інформаційному пошуку, машинному перекладі, автоматичному реферування та ін.

Призначення для NLP можна сформулювати дуже влучною фразою «Подолання розриву зв'язку між людиною і комп'ютером».

1.1.2 Найбільш часто досліджувані завдання в NLP

Основними завданнями обробки природної мови залишаються [3]:

синтаксичні – лематизація, морфологічна сегментація, визначення частин мови кожного слова, аналіз, парсинг, сегментація слів; *семантичні* – лексична семантика, машинний переклад, Named entity recognition (NER), розуміння та генерація природної мови, питання -відповідь, розуміння відносин між словами та словосполученнями, розв'язання лексичної багатоманітності; *дискурс* – автоматичне реферування, аналіз дискурсу, автоматичне сортування, визначення сюжетів та відношень слів між собою; *мовні* – розпізнавання і сегментація мови, перетворення тексту в мову.

1.1.3 Види і підходи семантичного аналізу тексту

Як викладено у статті [3], існують слідуючі підходи семантичного аналізу тексту: *статистичний підхід* – базується на підрахунку кількості слів у текстах і не потребує глибоких теоретичних знань в лінгвістиці, використовуються усереднені дані і виявлені закономірності; *кібернетичний* – об'єднує множину різноманітних математичних підходів, пов'язуючи комп'ютерну математику та штучний інтелект; *лінгвістичний підхід* до обробки природної мови складається з графоматичного розбиття тексту та виділення розділів, абзаців, речень, морфологічного – визначення характеристик та атрибутів окремого слова, синтаксичного – визначення характеристик підрядного зв'язку слова в словосполученнях, семантичного – пов'язаний із сурядним словосполученням тексту та його сенсу;

синтактико-семантичний аналіз – завданням блоку синтаксичного аналізу є представлення кожного речення заданого природномовного тексту у вигляді синтаксичного дерева (лексми у реченні з синтактико-семантичними відношеннями між ними); *морфологічний аналіз* – сутність процедури морфологічного аналізу полягає у приписуванні кожній мовній лексемі вхідного природномовного об'єкту відповідної змістової інформації та їх структурування у морфологічній таблиці; *символічний підхід* – базується на явному представленні знань, що здійснюється шляхом використання добре досліджених схем та алгоритмів (словники, формули, правила); *коннективістський підхід* – відповідає за обробку загальних моделей з використанням конкретних прикладів мовних явищ; *метод допоміжних векторів* – диференційний метод машинного навчання, що допомагає провести класифікацію слів за категоріями; *прихована марковська модель* – це така графічна система, у якій кожна вершина представляє собою випадкову змінну, що може набувати будь-якого значення (з певними ймовірностями) між кількома станами, породжуючи при цьому один з декількох можливих вихідних символів з переходом для кожного з них. Множина всіх можливих станів та унікальних символів може бути дуже великою. Ми можемо бачити вихідні дані, проте початкові стани системи є прихованими; *умовні випадкові поля* – ця модель використовується для передбачення стану змінної та базується на спостереженнях за цією змінною; *n-грамні моделі* – використовуються в основному для прогнозу. В основі моделі лежить теорія ймовірності. N-грамна модель обчислює вірогідність останнього слова N-грами, за умовою знання всіх попередніх.

1.1.4 Напрямки використання NLP

Комп'ютерна лінгвістика, в основі якої лежить обробка природної мови, виникла в Сполучених Штатах в 1950-х роках. Основною метою було використовувати комп'ютери для автоматичного перекладу текстів з іноземних мов на англійську мову. Так як комп'ютери можуть зробити арифметичні обчислення набагато швидше і точніше, ніж люди, то вважалося, що це тільки питання часу, перш ніж вони могли б також розпочати процес мови.

Перші результати не були достатньо схвальні і протягом багатьох наступних десятиліть значні зусилля науковців були спрямовані на розробку і покращення цього напрямку.

У зв'язку з бурхливим розвитком Інтернету та інших комп'ютерно-комунікаційних технологій, ОПМ набуває ще більшого значення.

Методи семантичного аналізу тексту та ітеративного пошуку широко використовуються в інформаційно-пошукових системах.

Різке зростання соціальної важливості призвело до збільшення інформаційних потоків. Обсяги електронних документів значно збільшились. У зв'язку з чим зросла потреба у дослідженні та розробці систем автоматичного сортування, виділення сюжетів, реферування великих документальних масивів і пошук дублікатів. Динамічний характер інформаційного попиту спричиняє потребу у постійному отриманні оперативної інформації для своєчасного прийняття правильних рішень. Особливістю таких документів є те, що вони мають вигляд неструктурованих текстів, обробка яких не існує у створених ПЗ.

Також не менш важливим фактом є те що NLP відіграє базову роль у розвитку AI та робототехніки. Аналіз відносять до розуміння мови, а синтез до генерації розумного наповненого змістом тексту. Вирішення цих питань надасть можливість покращати взаємдію людини з комп'ютером.

1.2 Огляд існуючого інструментарію Stanford Core NLP

1.2.1 Stanford Core NLP основні можливості

Стенфорд CoreNLP надає широкий набір інструментів для аналізу природної мови. Вона може надати базові форми слів, частини мови, аббревіатури, власні назви, нормалізовані дати, час і числові величини. Також ця бібліотека може розпізнати структуру речення у термінах фраз та залежності слів. Вказує яка фраза відноситься до тієї ж сутності, вказує настрій, а також розуміє цитати [4].

Вона містить такі модулі аналізу тексту (Анотатори) Додаток Д, які підтримують наступні мови таблиця 1.1.

Таблиця 1.1 – Анотатори і підтримка природних мов

Анотатор	Арабська	Китайська	Англійська	Французька	Німецька
Tokenize	✓	✓	✓	✓	✓
Sent. split	✓	✓	✓	✓	✓
Truecase			✓		
POS	✓	✓	✓	✓	✓
Lemma			✓		
Gender			✓		
NER		✓	✓		✓
RegexNER	✓	✓	✓	✓	✓
Parse	✓	✓	✓	✓	✓
Dep. Parse		✓	✓		
Sentiment			✓		
Coref.			✓		

1.2.2 Досягнення в цій галузі та можливі етапи розвитку

Стенфорд CoreNLP має слідуючий перелік можливостей:

- а) інтегрований набір інструментів для граматичного аналізу;
- б) швидкий та надійний аналіз довільного тексту;
- в) висока якість при аналізі тексту;
- г) підтримка основних мов (англійська, арабська, китайська, французька, німецька, іспанська);
- д) доступні інтерфейси для основних сучасних мов програмування;
- е) можливість працювати як простий веб-сервіс.

Stanford Core NLP надає глибокі можливості обробки природної мови та інструменти для вирішення основних проблем з обчислювальної лінгвістики, які можуть бути включені в ПЗ з технологічними потребами в обробці людської мови. Інструментарій Stanford Core NLP широко використовується у промисловості, у наукових колах та урядових структурах [4].

Спершу Stanford Core NLP була розроблена тільки для Java, але зараз її можна використовувати у Python, Ruby, Perl, Javascript, F#, та інших .NET і JVM мовах.

Stanford Core NLP існує у відкритому доступі, що сприяє більшому розвитку напрямку обробки природної мови.

1.2.3 Складність роботи з Natural Language Processing і недоліки існуючих бібліотек

Не зважаючи на те що найбільш досконалі автоматизовані системи з обробки природної мови досягли задовільних теоретичних результатів, всі вони знаходяться в експериментальній стадії на сьогоднішній день. Тому жодна з розроблених теорій не може заявляти свої права на повноту рішення.

Найбільш відомим недоліком існуючих NLP систем є відсутність єдиного підходу до аналізу текстової інформації. Іншим недоліком є те що нові системи часто використовують матеріали попередніх аналітичних систем з ціллю скоротити час розробки. Внаслідок чого похибка функціонування створеної аналітичної системи збільшується за рахунок похибки попереднього ПЗ.

Не менш важливим можна назвати використання різних формальних моделей представлення текстової інформації на різних етапах обробки. Що приводить до формування різних результатів при багаторазовому перетворенні текстової інформації.

Результатом таких недоліків можуть бути неоднакові результати парсингу однакових типів тексту, що значно ускладнює подальшу обробку отриманих результатів.

Приклад: Тобто, оцінити недоліки таких систем і точно визначити довірчий інтервал для результатів роботи інтелектуальних систем дуже важко. Також необхідна обов'язкова підготовка вхідної текстової інформації для подальшого її опрацювання. Це пов'язане з необхідністю виявлення присутніх даних закономірностей, а також компактного розміщення природно-мовної інформації з метою оптимізації часу виконання обробки.

1.3 Засоби зберігання і візуального представлення тексту

1.3.1 XMI (XML Metadata Interchange) стандарт OMG для обміну мета даними

XMI (XML Metadata Interchange) є стандартом OMG для використання мови розмітки (XML), який призначений для забезпечення стандартного способу обміну метаданими [5]. Зокрема, XMI допомагає програмістам використовуючи Unified Modeling Language (UML) обмінюватися моделями

даних. Крім того, ХМІ може використовуватися для обміну інформацією у сховищах даних. Фактично, формат ХМІ стандартизує будь-який набір метаданих та вимагає однакового відображення цих даних у різних галузях використання.

1.3.2 XML мова розмітки даних. Основні принципи і методи роботи

Extensible Markup Language (XML) використовується для опису даних. Стандарт XML являє собою гнучкий спосіб для створення інформаційних форматів в електронному вигляді для обміну структурованими даними через загальнодоступні та корпоративні мережі такі як Інтернет [6].

XML, офіційна рекомендація від World Wide Web Consortium (W3C), схожий на Hypertext Markup Language (HTML). Обидва XML і HTML містять розмітки символів для опису сторінки або вмісту файлу.

XML відомий як самоописуємий або самовизначальний формат, що означає – структура даних вбудовується разом з даними таким чином, що коли дані надходять, немає необхідності попередньо створювати структуру для зберігання даних; вона динамічно зрозуміла [5]. Формат XML може бути використаний будь-якою особою, групою осіб або компаній, які хочуть обмінюватися інформацією.

Основний будівельний блок документу XML являє собою елемент, який визначається за допомогою тега. Елемент має початковий і кінцевий тег. Всі елементи в документі XML містяться в зовнішньому елементі, відомому як корневий елемент. XML також підтримує вкладені елементи. Ця здатність дозволяє XML підтримувати ієрархічні структури. Імена елементів описують вміст елемента, а структура описує взаємозв'язок між елементами.

1.3.3 Діаграми UML як один із засобів візуального представлення тексту

Unified Modeling Language(UML) містить в собі стандартизовані візуальні позначення, які можуть бути використані для подання потрібної інформації у належному вигляді.

UML був задуманий з метою створення семантичного та синтаксичного багатомовного візуального моделювання для архітектури, проектування та реалізації складних програмних систем, як структурних так і поведінкових [7].

UML складається з різних типів діаграм. UML діаграми описують межі, структуру і поведінку системи та об'єкти в ній.

Основною ціллю даної дипломної роботи є парсинг тексту та зберігання його у форматі XML. Це дає можливість для його представлення у вигляді UML діаграм як сукупність класів та їх залежностей. Більш детальний опис ви зможете розглянути у дипломній роботі [2].

1.4 Вектор розвитку даного напрямку і його перспективи

Машинна обробка природної мови є одним з найголовніших та найбільш досліджуваних напрямків сучасності. Не зважаючи на те що це питання далеке від остаточного вирішення, постійна увага і вклад з боку науковців, компаній та Open Source Community дають безперервні результати які просувають досягнення у цій області на нові щаблі та відкривають широкі можливості для використання поточних результатів у повсякденному житті.

Одним з найяскравіших прикладів сьогодення є використання NLP для перекладу з однієї мови на іншу (Google Translate).

Також NLP можна використовувати для побудови онтологій. Наприклад цей підхід використовує дуже популярний на сьогодні сервіс “Amazon Alexa” який може відповідати на прості питання у реальному часі.

1.5 Висновки

У цьому розділі я розглянула основні технології, мови, бібліотеки та інструментарій для роботи з Natural Language Processing. Виконала їх порівняння з урахуванням недоліків та переваг.

Цей етап є невід’ємною частиною для подальшого розвитку дипломного проекту. Адже він дозволяє адекватно зробити оцінку та передбачити приблизні результати, враховуючи недоліки та переваги одного чи іншого підходу.

Саме у слідуючому розділі я збираюсь використовувати отримані дані з метою подальшого впровадження функціоналу у кваліфікаційній роботі.

2 РЕАЛІЗАЦІЯ РОЗПІЗНАВАННЯ ТЕКСТУ І СТВОРЕННЯ UML ДІАГРАМ

У данному розділі описані характерні особливості розробки програмного забезпечення з використанням бібліотеки Stanford Core NLP та розглянуті основні технології та мови використані для реалізації дипломного проекту.

Далі буде надан детальний опис механізму побудови UML діаграм після парсингу тексту. Ми розглянемо алгоритм для побудови спрямованого графа з UML елементами, а також обробку кожної вершини графу та конвертацію їх у UML сутності.

На заключному етапі ми розглянемо програмний механізм реалізації парсера тексту, а також інструменти, бібліотеки, та середовища розробки які були використані на етапі створення проекту.

2.1 Специфіка і особливості реалізації

2.1.1 Технології, мови програмування, середовище, а також чинники їх вибору

Java була вибрана як основна мова для написання програми тому що це найпопулярніша мова програмування на сьогодні, а також вона має найбільш розвинуту інфраструктуру та Open Source Community. Це дозволяє швидко знаходити вирішення нагальних проблем, а також використовувати велику кількість існуючих бібліотек. Але найвагомішим критерієм вибору стала крос-платформна робота додатку, що означає можливість використання програми на Windows, Linux, Mac та інших операційних системах. Окрім

Java в проєкті використовується мова розмітки даних XML та її різновид XMI, який зберігає дані у форматі UML для подальшого відображення.

Основою дипломного проєкту є бібліотека Stanford Core NLP, яка використовується для обробки природної мови. Ця бібліотека розпізнає текст, та присвоює частини мови кожному слову, такі як іменник, дієслово, прикметник тощо. У нашому конкретному випадку ми отримуємо текст у вигляді ієрархічного дерева (графу) залежностей [8]. Яке буде використано у наступних етапах парсингу та обробці даних.

Також в проєкті використовується інструмент JiBX. JiBX відомий своїм найшвидшим та найгнучкішим підходом для зв'язування Java коду з XML.

Середовищем для розробки проєкту є Eclipse та IntelliJ IDEA (Community). Це найбільш популярні IDE для розробки Java проєктів на сьогодні.

2.1.2 Технічні складності при роботі з бібліотекою Core NLP

На початку роботи з цією бібліотекою виникло декілька складностей, першою з них була проблема з пам'ятю, другою і найбільшою була складність з підключенням додаткових мов таких як російська та українська.

Перша проблема є результатом, того, що для розпізнавання великих текстів, необхідний великий обсяг виділеної пам'яті. Цю проблему вдалось вирішити у програмі Eclipse (Neon) та IntelliJ IDEA (Community) які були успішно сконфігуровані для роботи з проєктом у разі якщо ця бібліотека споживає об'єм пам'яті більше ніж 2 гігабайти. Вирішенням цієї проблеми було встановлення необхідного флагу `-Xmx2g` для віртуальної машини Java.

Проблема з українською та російською мовами так і не була вирішена, тому що бібліотека Stanford Core NLP не підтримує їх взагалі. Цю проблему

можна частково вирішити використовуючи RDRPOSTagger Software. Це ПЗ може допомогти отримати розмітку для частин мови (POS - модуль у цій бібліотеці), але це не вирішує проблему повністю, інші модулі так і залишаться не задіяними, що не дозволить у повній мірі розпізнавати текст на цих мовах.

Альтернативою для розпізнавання українською та російською, може бути переведення на англійську, з подальшим використанням всіх можливостей Core NLP, але цей підхід внесе багато помилок, якщо переведення на англійську мову буде виконуватися машинально.

2.1.3 Передбачуваний кінцевий результат та складності його досягнення

Важко уявити більш непередбачуваний результат, ніж при використанні бібліотек для розпізнавання природної мови. Мабуть абсолютно точного та передбачуваного результату не вдасться досягти ще довгий час. Таким чином бібліотека Core NLP не є виключенням.

Перше, з чим довелося зустрітись, була різниця у результатах розпізнавання тексту у різних версіях цієї бібліотеки. Друге, це величезний вплив знаків пунктуації на кінцевий результат. Навіть кома може змінити дерево залежностей, що має сенс якщо ця кома стоїть у правильному місці, але якщо ми маємо справу з не редагованим текстом, у якому міститься велика кількість синтаксичних помилок, то результат може бути зовсім непередбачуваним. Що означає додаткову перевірку текстів та редагування перед безпосереднім парсингом. На сьогодні автоматична синтаксична перевірка текстів у популярних текстових редакторах таких як MS Word, Pages, та інших не може називатися вдалою, тому для підготовки тексту на англійській мові я намагалася використовувати сервіс Grammarly.

Наприклад ви можете побачити результат роботи парсингу для куплету з поеми Роберта Фроста – «Зупинений нічним засніженим лісом» [9]. Цей результат представляє собою дерево залежностей між словами в куплеті.

Текст для перевірки:

My little horse must think it queer
To stop without a farmhouse near
Between the woods and frozen lake
The darkest evening of the year.

```
(ROOT
(S
(NP (PRP$ My) (JJ little) (NN horse))
(VP (MD must)
(VP (VB think)
(S
(NP (PRP it))
(VP (VB queer)
(S
(VP (TO To)
(VP (VB stop)
(PP (IN without)
(NP
(NP (DT a) (NN farmhouse))
(PP (IN near)
(PP (IN Between)
(NP
(NP (DT the) (NNS woods))
(CC and)
(NP (JJ frozen) (NN lake))))))
(NP
(NP (DT The) (JJS darkest) (NN evening))
(PP (IN of)
(NP (DT the) (NN year)))))))))
(. .)))
```

Тепер розглянемо результати парсингу для того ж куплету, але додамо декілька ком.

My little horse must think it queer,

To stop without a farmhouse near,
Between the woods and frozen lake,
The darkest evening of the year.

```
(ROOT
(S
(NP (PRP$ My) (JJ little) (NN horse))
(VP (MD must)
(VP (VB think)
(NP (PRP it) (NN queer))
(, ,)
(S
(VP (TO To)
(VP (VB stop)
(PP (IN without)
(NP
(NP (DT a) (NN farmhouse))
(PP (IN near)
(PRN (, ,)
(PP (IN Between)
(NP
(NP (DT the) (NNS woods))
(CC and)
(NP (JJ frozen) (NN lake))))))
(, ,))
(NP
(NP (DT The) (JJS darkest) (NN evening))
(PP (IN of)
(NP (DT the) (NN year))))))))))
(. .)))
```

Тепер розглянемо результат з додаванням ком, як можна побачити то цей результат парсингу ще дуже схожий на попередній але з деякими відмінностями у цій частині (NP (PRP it) (NN queer)), відповідно це буде мати вплив на побудову UML діаграми та на вміст її елементів.

2.2 Механізм побудови UML діаграм

2.2.1 Алгоритм вилучення класів та їх залежностей з результатів парсинга тексту засобами Core NLP

Кілька слів треба сказати про структури даних які були використані для репрезентації моделей у програмі. Такі структури як граф, орієнтований граф, дерева, та мішаний граф.

Граф – це сукупність об'єктів із зв'язками між ними.

Об'єкти розглядаються як вершини, або вузли графу, а зв'язки – як дуги, або ребра.

Орієнтований граф – граф ребрам якого присвоєно напрямок. Орієнтовані ребра називаються також дугами, а в деяких джерелах і просто ребрами.

Мішаний граф – граф, що містить як дуги, так і ребра.

Дерево – в інформатиці та програмуванні одна з найпоширеніших структур даних. Формально дерево визначається як скінченна множина T з однією або більше вершин, яка задовольняє наступним вимогам:

- а) існує один відокремлений вузол – корінь дерева.
- б) інші вузли (за винятком кореня) розподілені серед $m \geq 0$ непересічних множин $T_1 \dots T_m$ і кожна з цих множин, в свою чергу, є деревом. Дерева $T_1 \dots T_m$ мають назву піддерев даного кореня.

Також у додатку А наданий список найбільш часто використовуваних умовних позначень для частин мови у Core NLP [10].

Тепер розглянемо алгоритм дій, який використовується для вилучення класів, та їх залежностей, з дерева, яке Core NLP надає по завершенню парсингу.

- а) передати текст до бібліотеки Core NLP та отримати результат (дерево залежностей). Кожна гілочка цього дерева являє собою фразу, набір

фраз, або слово яке проанотоване за допомогою таких анотаторів як tokenize, ssplit, pos, lemma, parse;

б) коли результат парсингу вже отриманий слідуючим кроком буде виділення всіх іменників та займенників (NP (PRP it) (NN queer)) у окремий граф де вони будуть вершинами.

в) всі дієслова (VP (VB think)) у свою чергу теж виділяються до окремого графу, вони будуть ребрами;

г) на наступному кроці, ми виділяємо всі прикметники з отриманого дерева залежностей, та додаємо їх до вже виділених іменників в окремому графі;

д) на п'ятому кроці ми намагаємося виділити ті частини мови які залишились, та додати їх у якості ребер, або вершин, до окремого графу, в якому на цей момент вже містяться іменники, дієслова та прикметники;

е) в результаті цих дій на кроці шість ми отримуємо проміжний результат у вигляді графу, який у подальшому буде використаний для побудови ще одного графу в якому іменники будуть перетворені на класи, дієслова на характерні залежності, а прикметники будуть перетворені на властивості класів;

ж) побудова UML графу на основі проміжного графу.

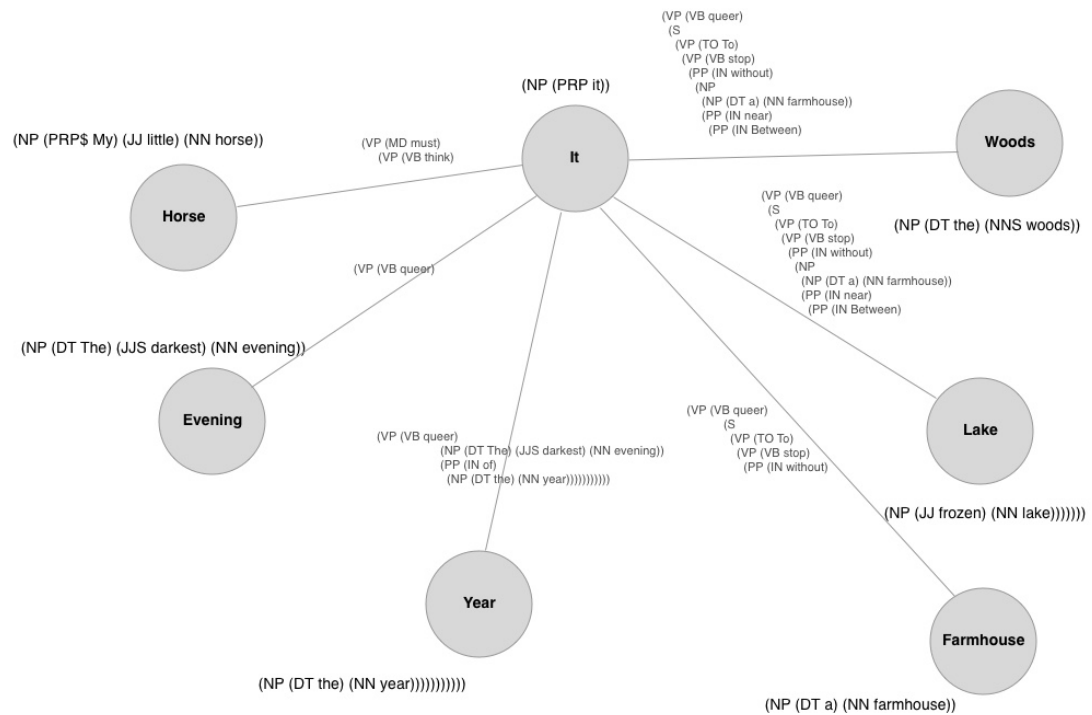


Рисунок 2.1 – Умовна схема зв'язків проміжного графу

Для проміжного графу рисунок 2.1 використовується текст

My little horse must think it queer

To stop without a farmhouse near

Between the woods and frozen lake

The darkest evening of the year.

2.2.2 Побудова проміжного графа та зазначення відносин між елементами

У цьому пункті ми детально розглянемо побудову проміжного графу рисунок. 2.1 на основі дерева залежностей отриманого від Core NLP.

Перше що було потрібно зробити, це визначити правила конвертації. Частково ці правила були створені на основі роботи [11], а також дуже допоміг документ [12].

Основні правила для конвертації можна представити у вигляді таблиці:

Таблиця 2.1 – Правила конвертації дерева Core NLP в проміжний граф

Вершини	Ребра
NP	VP

Нижче наведений результат виводу програми, при конвертації, для перевірки адекватності створених даних на основі правил у таблиці 2.1.

(NP (PRP\$ My) (JJ little) (NN horse)) —> (VP (VB think) (S (NP (PRP it)) (VP (VB queer) (S (VP (TO To) (VP (VB stop) (PP (IN without) (NP (NP (DT a) (NN farmhouse)) (PP (IN near) (PP (IN Between) (NP (NP (DT the) (NNS woods)) (CC and) (NP (JJ frozen) (NN lake)))))) (NP (NP (DT The) (JJS darkest) (NN evening)) (PP (IN of) (NP (DT the) (NN year))))))))) —> (NP (PRP it))

(NP (PRP it)) —> (PP (IN without) (NP (NP (DT a) (NN farmhouse)) (PP (IN near) (PP (IN Between) (NP (NP (DT the) (NNS woods)) (CC and) (NP (JJ frozen) (NN lake)))))) (VB queer)) —> (NP (DT a) (NN farmhouse))

(NP (PRP it)) —> (PP (IN without) (NP (NP (DT a) (NN farmhouse)) (PP (IN near) (PP (IN Between) (NP (NP (DT the) (NNS woods)) (CC and) (NP (JJ frozen) (NN lake)))))) (VB queer)) —> (NP (DT the) (NNS woods))

(NP (PRP it)) —> (PP (IN without) (NP (NP (DT a) (NN farmhouse)) (PP (IN near) (PP (IN Between) (NP (NP (DT the) (NNS woods)) (CC and) (NP (JJ frozen) (NN lake)))))) (VB queer)) —> (NP (JJ frozen) (NN lake))

(NP (PRP it)) —> (VB queer) —> (NP (DT The) (JJS darkest) (NN evening))

(NP (PRP it)) —> (PP (IN of) (NP (DT the) (NN year)) (VB queer)) —> (NP (DT the) (NN year))

Дивлячись на результати конвертації, можна побачити що було створено декілька вершин та ребер, вершини знаходяться до і після стрілочок, а ребра між ними.

Тепер коли ми маємо проміжний граф ми можемо переходити до створення UML графу з наступним перетворенням цього графу у UML діаграму.

2.2.3 Створення UML графу та пост обробка кожного елемента графа з перетворенням в UML елементи

На данному етапі ми вже маємо створений проміжний граф з залежностями між словами та фразами. Першим кроком буде визначення правил для конвертації з проміжного графу. Ці правила були визначені самостійно і можуть бути змінені у майбутньому.

Таблиця 2.2 – Правила конвертації проміжного графу у UML граф

Назва частини мови	UML сутності
NN, NNP, PRP, NNS	Класи
JJ, CD, RB	Атрибути класів
VBP, VBN, VBG, IN, TO, VBZ, ADVP, VB	Залежності між класами (асоціація, агрегація, генералізація)
ADJP, PP, SBAR	Додаткова інформація для залежностей між класами, яка впливає на те як вони будуть конвертовані
IN	вказує на агрегацію або генералізацію в залежності від контексту
CC	Поєднання однакових по типу залежностей

Після того як правила будуть застосовані до даних з проміжного графу, програма виведе наступне:

Horse —> think queer To stop without near Between of —> It
 It —> without near Between queer —> Farmhouse
 It —> without near Between queer —> Woods

It —> without near Between queer —> Lake
 It —> queer —> Evening
 It —> of queer —> Year

Як і у попередньому графі, до і після стрілочок виділені класи (вершини), а зв'язки (ребра) знаходяться посередені. Надані результати у цьому конкретному випадку можуть виглядати сумнівними, але після конвертації у UML діграму, вони будуть мати сенс.

Також розглянемо інший приклад для тексту:

Cats are similar in anatomy to the other fields, with a strong flexible body, quick reflexes, sharp retractable claws, and teeth adapted to killing small prey.

Cats —> similar in are —> Anatomy
 Cats —> to are —> Fields
 Cats —> with are —> Body
 Cats —> with are —> Reflexes
 Cats —> with are —> Claws
 Teeth —> to killing —> Prey

У цьому тексті ми можемо бачити вже набагато зрозуміліший результат, у разі якщо ми використовуємо прості речення які не є віршованими.

Тепер давайте розглянемо UML діграми які будуть отримані після приведення UML графу до формату XMI. Як візуальний редактор для роботи та аналізу UML, був обраний ArgoUML [13], також була використана модифікована версія цього редактора з кваліфікаційної роботи [2]

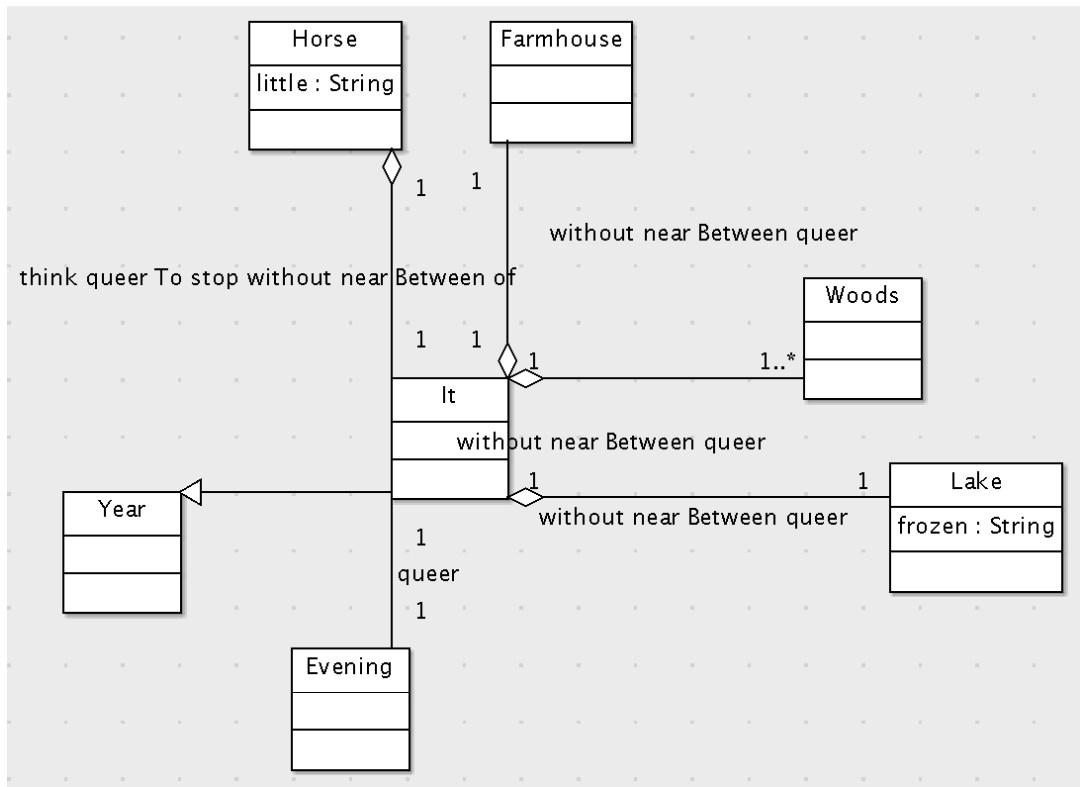


Рисунок 2.2 – UML діаграма для куплету з поеми [9]

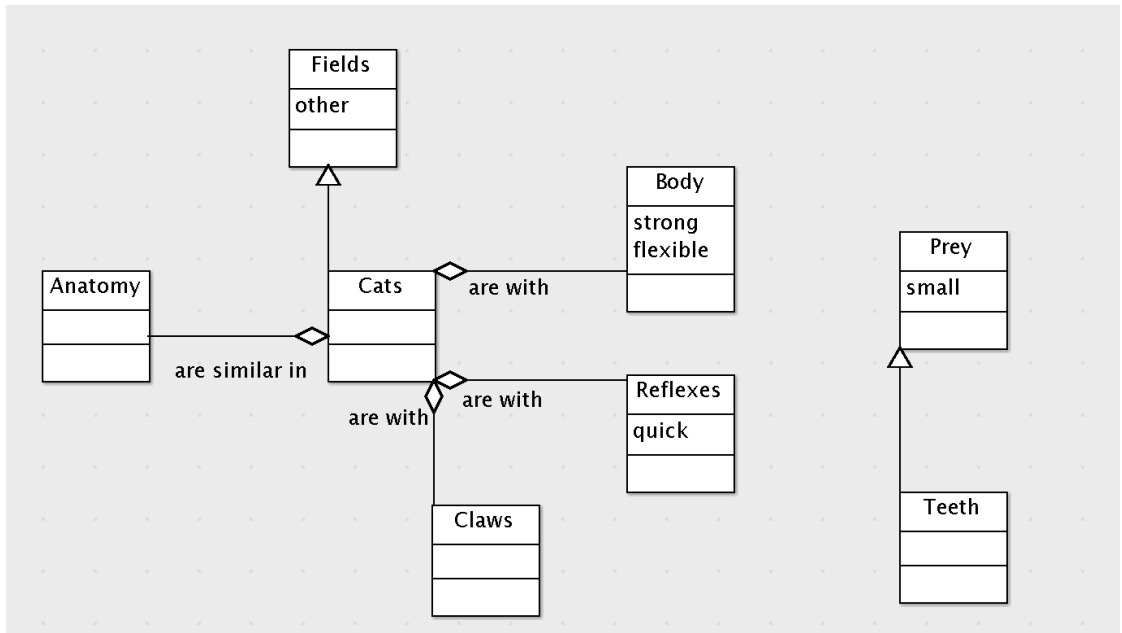


Рисунок 2.3 – UML діаграма для тексту «Cats are similar in anatomy...»

2.3 Робота з форматом ХМІ

2.3.1 Впровадження та використання `java.xml` пакета із стандартного набору розширень Java

Для того щоб із UML графу отримати UML діаграму, мати можливість провести аналіз та редагувати результати парсингу, нам необхідно конвертувати UML граф у формат ХМІ та зберегти його як файл з розширенням `.xmi`

Для запису в `.xmi` файл використовується стандартний набір розширень `java.xml` (пакет класів) та JAXB фреймворк.

Java Architecture for XML Binding (JAXB) програмне забезпечення, яке дозволяє розробникам Java відображати класи Java як XML репрезентації. JAXB має дві основні функції, здатність конвертувати об'єкти Java в XML, а також відновлювати такі об'єкти з XML [15].

2.3.2 Перетворення класів в ХМІ об'єкти

За допомогою фреймворку JAXB вдається автоматично конвертувати класи в ХМІ сутності, наприклад клас `Horse` який буде міститись у UML графу поеми [9] буде виглядати так

```
<UML:Class visibility="public" isActive="false" xmi.id="Horse_ClassID0"
name="Horse" isSpecification="false" isRoot="false" isLeaf="false"
isAbstract="false">
  <UML:Classifier.feature>
    <UML:Attribute xmi.id="little0" name="little" visibility="private"
isSpecification="false" ownerScope="instance">
      <UML:StructuralFeature.type>
        <UML:DataType href="http://argouml.org/profiles/uml14/default-
uml14.xmi#-84-17--56-5-43645a83:11466542d86:-8000:0000000000000087E"/>
      </UML:StructuralFeature.type>
```

```

</UML:Attribute>
</UML:Classifier.feature>
</UML:Class>

```

Як можна побачити у наведеному коді клас Horse містить властивість little типу String.

```

<UML:Attribute xmi.id="little0" name="little" visibility="private"
isSpecification="false" ownerScope="instance">
  <UML:StructuralFeature.type>
    <UML:DataType href="http://argouml.org/profiles/uml14/default-
uml14.xmi#-84-17--56-5-43645a83:11466542d86:-8000:000000000000087E"/>
  </UML:StructuralFeature.type>
</UML:Attribute>

```

Це означає що перетворення об'єкту класа у XMI сутність пройшло вдало. Наступним прикладом буде перетворення зв'язків між класами у XMI елементи. Розглянемо XML код асоціації між класом Horse та It додаток В.

Цей XML код є результатом роботи JAXB, а це означає що зв'язки у UML графі репрезентовані спеціальними об'єктами класів, котрі конвертуються у необхідний тип зв'язку UML. Ці класи були створені у роботі [14].

2.4 Програмний механізм реалізації

2.4.1 Впровадження бібліотеки CoreNLP

У данному проекті використовується декілька сторонніх бібліотек з відкритим кодом окрім Core NLP. Так як ручний процес інтеграції цих бібліотек у проект може бути доволі працемістким, було вирішено скористатися менеджером залежностей Apache Maven.

Apache Maven є інструментом управління програмним продуктом. На основі концепції об'єкта проекту моделі (POM), Maven може централізовано керувати збіркою, звітністю та документацією проекту.

Для того щоб використати Apache Maven нам портібно:

- а) створити проект Java на основі Maven;
- б) вказати у створеному файлі pom.xml необхідні залежності, бібліотеки, фреймворки, тощо. Приклад цього файлу буде наведений нижче.

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.svitlanamoiseyenko</groupId>
  <artifactId>ontology.text-to-uml.parser</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>Text to UML parser</name>
  <description>The Development of Software for Parsing Texts and Generating
their UML Models</description>

  <dependencies>
    <dependency>
      <groupId>edu.stanford.nlp</groupId>
      <artifactId>stanford-parser</artifactId>
      <version>3.7.0</version>
      <scope>compile</scope>
      <classifier>models</classifier>
    </dependency>
    <dependency>
      <groupId>org.jgrapht</groupId>
      <artifactId>jgrapht-core</artifactId>
      <version>1.0.1</version>
    </dependency>
  </dependencies>
</project>
```

Елемент перший dependency відповідає за підключення Core NLP бібліотеки. Все що потрібно було зробити для імпортування, це взяти на сайті Stanford Core NLP необхідний код і додати його до pom.xml, далі потрібно виконати команду `mvn install`, але у моєму випадку я використовувала IDE IntelliJ IDEA яка вже містить необхідний функціонал, для автоматичного додавання бібліотек після необхідних змін у pom.xml файлі. Також це можна зробити за допомогою меню Maven → Reimport.

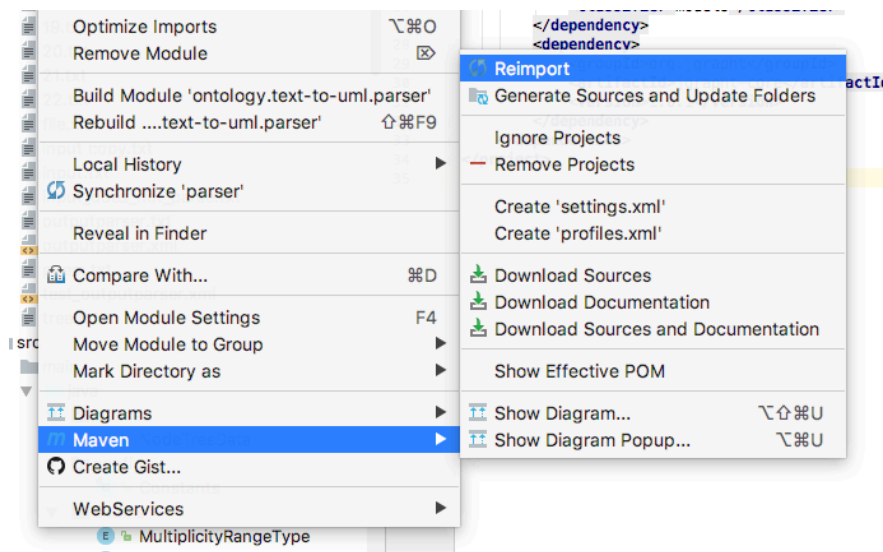


Рисунок 2.4 – Меню інтегрованого плагіна Apache Maven у IntelliJ IDEA

2.4.2 Отримання результатів парсинга у вигляді дерева сутностей (NP, VP ...)

Для отримання результатів парсингу, у вигляді дерева залежностей між частинами мови, як було зазначено у попередніх розділах, я використала бібліотеку Core NLP. Нижче наведені приклади коду, короткий опис класів та їх використання.

Клас `TextParser` являє собою модель парсера текста, який виконує основну роботу по управлінню процесами та етапами створення графів,

можна відноситись до нього як до сценарію перетворення текста природною мовою на XMI об'єкт для подальшого запису у файл .xmi.

```
public class Main {

    public static void main(String[] args) {

        TextParser textParser = new TextParser();
        XMI xmi = textParser.process("poem.txt");
        XMI_output.WriteToXMIFile(Constants.resourcesDir + "test_outputparser" +
        ".xmi", xmi);
    }
}
```

Кожна програма Java починається з класу Main та статичного методу з такою ж назвою. Перше що ми робимо це створюємо об'єкт типу TextParser, далі буде викликаний метод process, у який буде передано назву текстового файлу poem.txt, який містить текст для парсингу. Тепер розглянемо детальніше код методу process.

```
public XMI process(String filename) {
    String fileText = FileManager.readFile(filename);

    Properties props = new Properties();
    props.put("annotators", "tokenize, ssplit, pos, lemma, parse");
    pipeline = new StanfordCoreNLP(props);

    System.out.println("pipeline: " + pipeline);
    Annotation document;
    document = new Annotation(fileText);
    pipeline.annotate(document);
    System.out.println(" pipeline annotation" + document);
    XMI xmiStructure = null;

    ParseDocument(document);
    //buildAbstractModelElementsList();
    xmiStructure = buildXMI(xmiHelper.abstractModelElements);

    return xmiStructure;
}
```

Тут створюються властивості, які будуть використані у подальшому для конфігурування бібліотеки Core NLP. Безпосередньо, створюється властивість для анотаторів які в свою чергу параметризують механізм парсингу цієї бібліотеки.

Наступним створюється об'єкт типу Annotation в який ми передаємо текст з прочитаного файлу, з цього об'єкта викликається метод annotate, у який ми передаємо об'єкт анотованого документу. Далі цей же анотований документ ми передаємо у метод ParseDocument для подальших перетворень результатів роботи Core NLP у проміжний граф.

2.4.3 Побудова проміжного графа на основі отриманого дерева

Розглянемо програмну побудову проміжного графу на основі отриманого дерева залежностей. Почнемо з методу ParseDocument.

```
private void ParseDocument(Annotation document) {
    List<CoreMap> sentences = document.get(SentencesAnnotation.class);
    for (CoreMap sentence : sentences)
    {
        // this is the parse tree of the current sentence
        Tree tree = sentence.get(TreeAnnotation.class);
        System.out.println("tree: ");
        TreePrint treePrint = new TreePrint("penn");// latexTree
        treePrint.printTree(tree);

        buildNPGraph(tree);
        buildXMIGraph();
        buildAbstractModelElementsList();
    }
}
```

Перше що робить метод ParseDocument, це обхід кожного об'єкту речення у документі, та створює дерево залежностей між фразами та словами

цього дерева. Наступним кроком виводимо це речення до консолі рисунок 2.5 у вигляді Latex Tree за допомогою метода printTree який міститься у об'єкті типу TreePrint з бібліотеки Core NLP.

```

Run Main
My little horse must think it queer
To stop without a farmhouse near
Between the woods and frozen lake
The darkest evening of the year.
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
pipeline: edu.stanford.nlp.pipeline.StanfordCoreNLP@5f282abb
pipeline annotationMy little horse must think it queer
To stop without a farmhouse near
Between the woods and frozen lake
The darkest evening of the year.
tree:
(ROOT
  (S
    (NP (PRP$ My) (JJ little) (NN horse))
    (VP (MD must)
      (VP (VB think)
        (S
          (NP (PRP it))
          (VP (VB queer)
            (S
              (VP (TO To)
                (VP (VB stop)
                  (PP (IN without)
                    (NP
                      (NP (DT a) (NN farmhouse))
                      (PP (IN near)
                        (PP (IN Between)
                          (NP
                            (NP (DT the) (NNS woods))
                            (CC and)
                            (NP (JJ frozen) (NN lake))))))
                    (NP
                      (NP (DT The) (JJ$ darkest) (NN evening))
                      (PP (IN of)
                        (NP (DT the) (NN year))))))))))
            (NP
              (NP (DT The) (JJ$ darkest) (NN evening))
              (PP (IN of)
                (NP (DT the) (NN year))))))))))
    (NP
      (NP (DT The) (JJ$ darkest) (NN evening))
      (PP (IN of)
        (NP (DT the) (NN year))))))
  )
)
  
```

Рисунок 2.5 – Консоль IntelliJ IDEA та виведення дерева у вигляді Latex Tree

Після виводу дерева залежностей, викликається метод buildNPGraph для побудови проміжного графу. Розглянемо цей метод та два інших метода buildGraphNodes та buildGraphEdges

```

private void buildNPGraph(Tree tree) {
    buildGraphNodes(tree);
    buildGraphEdges(tree);
}
  
```

```

private List<Tree> buildGraphNodes(Tree tree) {
    List<Tree> nodesNP = new ArrayList<Tree>();
    treeManager.getNPTrees(tree, nodesNP);
    npGraph.addNodes(nodesNP);

    return nodesNP;
}

private void buildGraphEdges(Tree tree) {
    System.out.println("buildGraphEdges: ");
    List<NPNode> graphNodes = npGraph.getAllNodes();
    for (NPNode node: graphNodes)
    {
        NPNode parentNode = node;
        List<NodeTreeData> connectedItems =
            new ArrayList<NodeTreeData>();
        treeManager.getRelatedNP(tree, parentNode.tree, tree,
connectedItems);
        npGraph.addEdges(parentNode, connectedItems);
    }

    npGraph.printGraph();
}

```

Метод обгортка buildNPGraph викликає два методи buildGraphNodes та buildGraphEdges які додають вершини та ребра до проміжному графу. Для побудови графів у програмі використовується бібліотека JGraphT.

JGraphT є вільною для використання Java бібліотекою, яка надає математичну теорію графів та алгоритми для її використання.

Далі методи getNPTrees та getRelatedNP об'єкта TreeManager викликають одноіменні методи класу BaseTreeUtil.

TreeManager являє собою клас «одинак» який може існувати тільки у одному екземплярі, також цей клас було оптимізовано для використання у багатопотоковому середовищі, так як планувалося робити обробку дерев паралельно, що може значно скоротити час обробки великих текстів.

BaseTreeUtil клас є частиною пакету utils.tree цей пакет потрібен для роботи з деревами CoreNLP а також для конвертації цих дерев у спеціалізовані дерева зв'язків між іменниками та іншими частинами мови для проміжного графу.

У цих методах дотаток Г. відбувається основна робота з відокремлення частин мови та створення вершин і ребер, базуючись на правилах з таблиця 2.1. Після того як дані будуть оброблені та додані до проміжного графу, він буде переданий для подальшої конвертації у UML граф. Це реалізовано викликом метода buildXMIGraph з класу TextParser. Але перед тим як перейти до створення UML графу, у додатку Б розглянемо декілька констант з класу Constants, які потрібні для відтворення правил з таблиця 2.1 і таблиця 2.2 у коді програми.

2.4.4 Побудова UML діаграми шляхом перетворення елементів базового графа

Побудова UML графу розпочинається після того як метод buildNPGraph збудує проміжний граф і програма буде готова до побудови UML графу, це буде зроблено після виклику метода buildXMIGraph код якого наведений нижче.

```
private void buildXMIGraph() {
    List<NPNode> graphNodes = npGraph.getAllNodes();
    for (NPNode node: graphNodes) {
        xmiGraph.addNode(XMIGraphUtil.getXMINode(node));
    }
    List<NPEdge> graphEdges = npGraph.getAllEdges();
    for (NPEdge edge: graphEdges) {
        xmiGraph.addEdge(XMIGraphUtil.getXMIEdge(edge));
    }
    xmiGraph.printGraph();
}
```

Задачею цього метода є обробка кожної вершини і ребра проміжного графу та перетворення їх у вершини і ребра UML графу. Також тут використовується клас XMIGraphUtil з пакету utils.xmi. Призначенням цього пакету є конвертація сутностей проміжного графу за правилами з таблиця 2.2 у сутності UML графу.

Наступним кроком після побудови UML графу буде генерація UML сутностей, виклик buildAbstractModelElementsList побудує список UML елементів, таких як класи, зв'язки типа генералізація, асоціація та інші. Розглянемо код цього метода:

```
private void buildAbstractModelElementsList() {
    List<XMNode> graphNodes = xmiGraph.getAllNodes();

    for (XMNode node: graphNodes) {
        Class element = xmiHelper.getClassElementFromNode(node);
        xmiHelper.addElementToClass(element);
    }

    List<XMEdge> graphEdges = xmiGraph.getAllEdges();

    for (XMEdge edge: graphEdges) {
        XMNode parentNode = xmiGraph.getNode(edge.parentNodeId);
        XMNode childNode = xmiGraph.getNode(edge.childNodeId);

        AbstractModelElement abstractModelElement = xmiHelper
            .getConnectionElement(edge, parentNode, childNode);
        xmiHelper.addElementToClass(abstractModelElement);
    }
}
```

У цьому методі вилучаються усі вершини та ребра з UML графу та перетворюються на XMI об'єкти за допомогою класу AbstractModelElement з пакету legacy.xmi.model. Після чого використовується XMIUtil клас з пакету utils.xmi для створення списку з UML елементами які будуть готові до запису у .xmi файл.

AbstractModelElement в пакеті legacy.xmi.model є запозиченим класом з роботи [26].

XMIUtil клас з пакету utils.xmi імплементує логіку для роботи з вершинами та ребрами UML графу, ця логіка полягає в тому, що кожна вершина та ребро буде конвертоване у необхідну UML сутність, яка буде розпізнана у подальшому JAXB фреймворком.

При проектуванні цих класів та моделей даних, була розглянута можливість використовувати тільки проміжний граф який містить іменники та їх зв'язки між собою, а також мати можливість одразу конвертувати цей граф у XMI сутності з подальшим записом у файл. Але така можливість була відкинута на користь допоміжному UML графу. У разі використання тільки проміжного графу механізм конвертації був би набагато складніший ніж при використанні обох графів. Так як вершини і ребра проміжного графу містять дерева типу Tree, така структура даних буде дуже важко конвертуватись у список XMI об'єктів.

2.4.5 Генерація XMI файлу з метою зберігання та подальшої передачі даних

Після того як був отриманий проміжний та UML графи, а також дані з цих графів були конвертовані у список XMI елементів, переходимо до запису у файл формату .xmi. Розглянемо код з якого починається запис у файл.

```
public static void main(String[] args) {
    TextParser textParser = new TextParser();
    XMI xmi = textParser.Processing("poem.txt");
    XMI_output.WriteToXMIFile(Constants.resourcesDir + "test_outputparser" +
    ".xmi", xmi);
}
```

Тут виконується виклик статичного метода WriteToXMIFile з класу XMI_output до якого передається назва файлу для запису, а також отримані дані у вигляді списку XMI елементів. Розглянемо детальніше код методу WriteToXMIFile.

```
public static void WriteToXMIFile(String _filepath,XMI _xmi) {
    try {
        File file = new File(_filepath);
        JAXBContext jaxbContext =
JAXBContext.newInstance(XMI.class);
        Marshaller jaxbMarshaller = jaxbContext.createMarshaller();

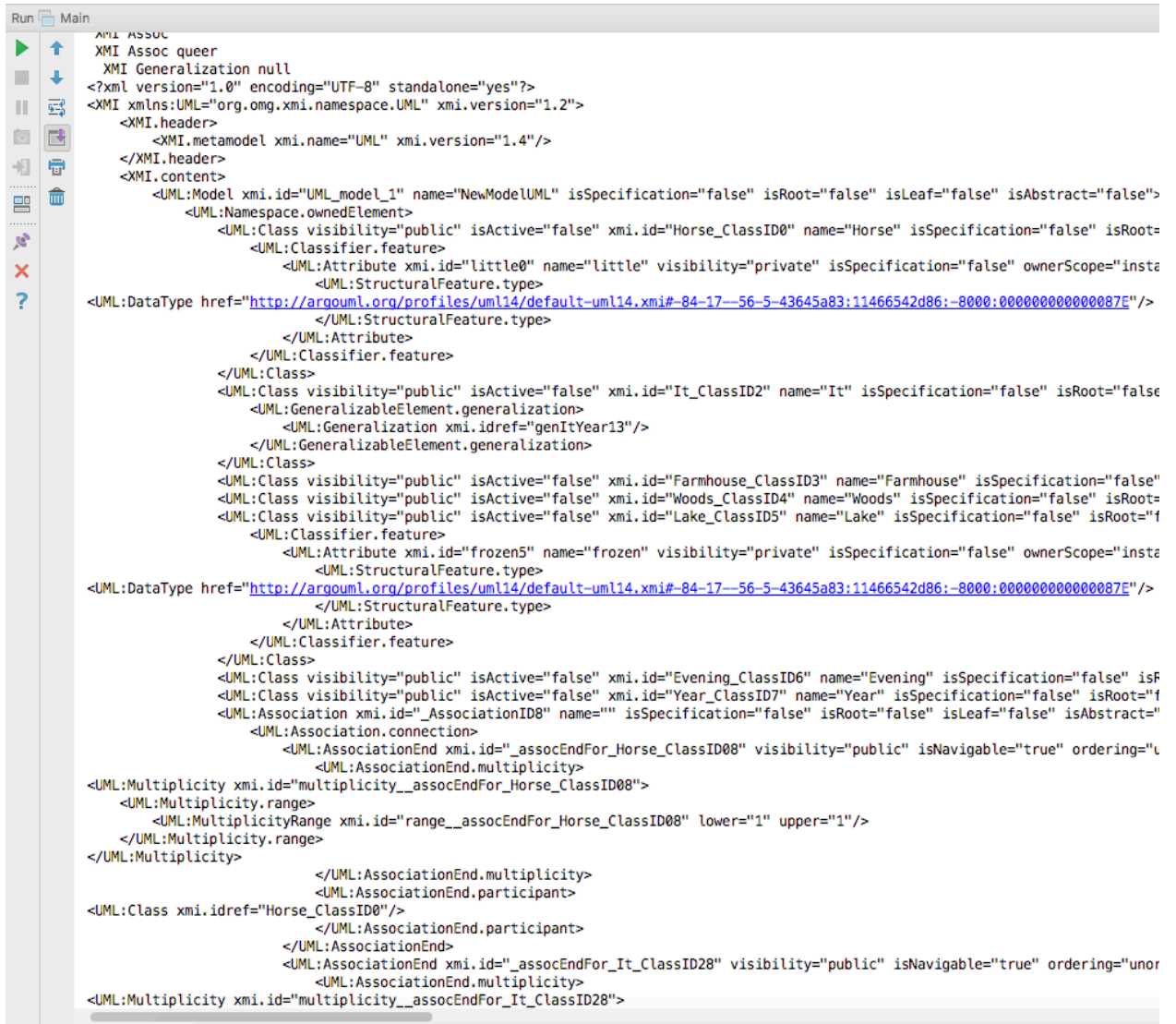
        // output pretty printed

        jaxbMarshaller.setProperty(Marshaller.JAXB_FORMATTED_OUTPUT,
true);

        jaxbMarshaller.marshal(_xmi, file);
        jaxbMarshaller.marshal(_xmi, System.out);

    } catch (JAXBException e) {
        e.printStackTrace();
    }
}
```

Спочатку створюється об'єкт файлу, до якого буде відбуватися запис, потім створюється об'єкт типу Marshaller з фреймворку JAXB, його призначенням є виконання запису XMI об'єкта (маршалінгу). Наступним буде конвертовано XMI об'єкт у XML код який буде записано до .xmi файлу. Після того як запис відбудеться, програма парсер завершує свою роботу, та виводить XML код до консолі (рис. 2.6).



```

Run Main
XMI ASSOC
XMI Assoc queer
XMI Generalization null
<?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="Horse_ClassID0" name="Horse" isSpecification="false" isRoot="false" isLeaf="false" isAbstract="false">
          <UML:Classifier.feature>
            <UML:Attribute xmi.id="little0" name="little" visibility="private" isSpecification="false" ownerScope="instance" isReadOnly="false">
              <UML:StructuralFeature.type>
                <UML:DataType href="http://argouml.org/profiles/uml14/default-uml14.xmi#-84-17-56-5-43645a83:11466542d86:-8000:00000000000087E"/>
              </UML:StructuralFeature.type>
            </UML:Attribute>
          </UML:Classifier.feature>
        </UML:Class>
        <UML:Class visibility="public" isActive="false" xmi.id="It_ClassID2" name="It" isSpecification="false" isRoot="false" isLeaf="false" isAbstract="false">
          <UML:GeneralizableElement.generalization>
            <UML:Generalization xmi.idref="genItYear13"/>
          </UML:Generalization>
        </UML:Class>
        <UML:Class visibility="public" isActive="false" xmi.id="Farmhouse_ClassID3" name="Farmhouse" isSpecification="false" isRoot="false" isLeaf="false" isAbstract="false">
        </UML:Class>
        <UML:Class visibility="public" isActive="false" xmi.id="Woods_ClassID4" name="Woods" isSpecification="false" isRoot="false" isLeaf="false" isAbstract="false">
        </UML:Class>
        <UML:Class visibility="public" isActive="false" xmi.id="Lake_ClassID5" name="Lake" isSpecification="false" isRoot="false" isLeaf="false" isAbstract="false">
          <UML:Classifier.feature>
            <UML:Attribute xmi.id="frozen5" name="frozen" visibility="private" isSpecification="false" ownerScope="instance" isReadOnly="false">
              <UML:StructuralFeature.type>
                <UML:DataType href="http://argouml.org/profiles/uml14/default-uml14.xmi#-84-17-56-5-43645a83:11466542d86:-8000:00000000000087E"/>
              </UML:StructuralFeature.type>
            </UML:Attribute>
          </UML:Classifier.feature>
        </UML:Class>
        <UML:Class visibility="public" isActive="false" xmi.id="Evening_ClassID6" name="Evening" isSpecification="false" isRoot="false" isLeaf="false" isAbstract="false">
        </UML:Class>
        <UML:Class visibility="public" isActive="false" xmi.id="Year_ClassID7" name="Year" isSpecification="false" isRoot="false" isLeaf="false" isAbstract="false">
        </UML:Class>
        <UML:Association xmi.id="_AssociationID8" name="" isSpecification="false" isRoot="false" isLeaf="false" isAbstract="false">
          <UML:Association.connection>
            <UML:AssociationEnd xmi.id="_assocEndFor_Horse_ClassID08" visibility="public" isNavigable="true" ordering="1">
              <UML:AssociationEnd.multiplicity>
                <UML:Multiplicity xmi.id="multiplicity__assocEndFor_Horse_ClassID08">
                  <UML:Multiplicity.range>
                    <UML:MultiplicityRange xmi.id="range__assocEndFor_Horse_ClassID08" lower="1" upper="1"/>
                  </UML:MultiplicityRange>
                </UML:Multiplicity>
              </UML:AssociationEnd.multiplicity>
            </UML:AssociationEnd>
            <UML:AssociationEnd xmi.id="_assocEndFor_It_ClassID28" visibility="public" isNavigable="true" ordering="unorderable">
              <UML:AssociationEnd.multiplicity>
                <UML:Multiplicity xmi.id="multiplicity__assocEndFor_It_ClassID28">
                  <UML:Multiplicity.range>
                    <UML:MultiplicityRange xmi.id="range__assocEndFor_It_ClassID28" lower="1" upper="1"/>
                  </UML:MultiplicityRange>
                </UML:Multiplicity>
              </UML:AssociationEnd.multiplicity>
            </UML:AssociationEnd>
          </UML:Association.connection>
        </UML:Association>
      </UML:Namespace.ownedElement>
    </UML:Model>
  </XMI.content>
</XMI>

```

Рисунок 2.6 – Вивід XML (XMI) коду до консолі в IntelliJ IDEA

Для перевірки на валідність XML (XMI) коду було використано онлайн валідатор xmlvalidation.com. А також найкращою перевіркою є те що .xmi файл може бути відкритий у програмі ArgoUML.

2.5 Переваги і недоліки

Розглянемо таблицю переваг та недоліків на архітектурному рівні, а також у програмному коді.

Переваги:

- а) застосування бібліотеки Stanford CoreNLP дає широкі можливості виконання різних типів парсингу;
- б) обробка отриманого дерева через виділення основних вузлів NP та VP без чіткої прив'язки до кожного типу залежності;
- в) використання JgraphT графу як основної структури даних;
- г) швидка обробка невеликих об'ємів текстів;
- д) використання менеджера залежностей Maven. Цей менеджер дозволяє підключати будь-які бібліотеки які знаходяться у відкритому доступі.

Недоліки:

- а) Stanford CoreNLP може надавати непередбачувані результати парсингу, це залежить від багатьох факторів наприклад, помилки в тексті, неправильно поставлені знаки пунктуації та інші;
- б) так як обробка текстів виконується не паралельно, парсинг великого об'єму тексту може займати досить багато часу. У майбутньому треба провести оптимізацію цього процесу;
- в) поточний підхід до конвертації дерева залежностей між частинами мови хоч і є універсальним, але не є досить надійним. В окремих випадках ми можемо отримати досить непередбачувані результати, які можуть дуже сильно відрізнятись від очікуваних;
- г) на теперішній час досі не існує чітких правил конвертації тексту, тому досить важко вивести успішну формулу для конвертації природної мови до UML діаграми;
- д) дуже велика залежність від результатів парсингу Stanfrod Core NLP. У майбутньому можна розглянути альтернативне або комплексне

рішення (наприклад використання додаткових бібліотек чи оптимізацію Core NLP).

2.6 Висновки

У цьому розділі описана специфіка і особливості реалізації програми парсера, технології, мови, та середовища розробки які були використані у процесі розробки. Був розглянутий механізм побудови UML діаграм, описані алгоритми для вилучення класів з дерева залежностей яке надає Core NLP. Також у цьому розділі були розроблені правила для конвертації частин мови у UML сутності. У розділі 2.3 розглядається робота з форматом XMI, впровадження та використання пакету `java.xml` для запису XMI сутностей у файл. У розділі 2.4 описана логіка роботи програми та основні архітектурні особливостей парсеру, також у цьому розділі описано впровадження бібліотеки Core NLP та її використання. Також була наведена логіка побудови UML діаграми шляхом перетворення елементів проміжного графу у UML елементи.

Результатом виконаної роботи є розроблений програмний продукт для парсингу тексту, який використовує результат роботи бібліотеки CoreNLP, а саме – дерево залежностей, виконує його парсинг згідно розробленим правилам. Далі на основі отриманих даних, генерує UML модель та зберігає результат у форматі XMI.

Даний результат буде використовуватись у якості вхідних даних у кваліфікаційній роботі Олександра Василейко [2] з ціллю подальшого проведення їх аналізу, редагування та конвертації у формат OWL. Розроблене програмне забезпечення не є досконалим, тому що воно не вирішує всіх пролем з якими можна зіткнутися у процесі його роботи. Тому наступним кроком буде проведення експеременту та визначення всіх недоліків та переваг з метою покращення.

3 ЕКСПЕРИМЕНТАЛЬНА ПЕРЕВІРКА РЕЗУЛЬТАТІВ РОБОТИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Мета експерименту

Метою експерименту є перевірка конвертації тексту природною мовою у UML діаграму та подальший аналіз отриманої діаграми. Потрібно відразу зазначити, що не існує чітких прийнятих правил для такого виду конвертацій. Тому будуть використані правила з таблиця 2.2.

3.2 План і проведення експерименту

Перевірка розробленого програмного забезпечення буде проведена у чотири етапи. На першому та другому етапах буде проведене тестове розпізнавання семантично насичених текстів малої довжини. На третьому етапі буде проведений тестовий парсинг тексту великої довжини. На четвертому етапі буде проведене розпізнавання та конвертація тексту з помилками як орфографічними так і синтаксичними. Після парсингу та конвертації текста будуть створені UML діаграми.

Такий план дозволить провести перевірку програмного продукту, у умовах використання, наближених до реальних. А також гарантує те, що будуть виявлені слабкі сторони як архітектурних так і програмних рішень даного ПЗ.

3.2.1 Вхідні дані

У якості вхідних даних будуть використовуватись такі тексти

а) куплет з поеми [9]

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

б) короткий текст опису котів як хижих тварин, взятий з вікіпедії.

Cats are similar in anatomy to the other fields, with a strong flexible body, quick reflexes, sharp retractable claws, and teeth adapted to killing small prey.

в) Опис компанії Apple.

Apple was founded by Steve Jobs, Steve Wozniak, and Ronald Wayne in April 1976 to develop and sell personal computers. It was incorporated as Apple Computer, Inc. in January 1977, and was renamed as Apple Inc. in January 2007 to reflect its shifted focus toward consumer electronics. Apple joined the Dow Jones Industrial Average in March 2015.

Apple is the world's largest information technology company by revenue, the world's largest technology company by total assets, and the world's second-largest mobile phone manufacturer, by volume, after Samsung. In November 2014, Apple became the first U.S. company to be valued at over US\$700 billion in addition to being the largest publicly traded corporation in the world by market capitalization. The company employs 115,000 full-time employees as of July 2015 and maintains 478 retail stores in seventeen countries as of March 2016. It operates the online Apple Store and iTunes Store, the latter of which is the world's largest music retailer. Consumers use more than one billion Apple products worldwide as of March 2016.

г) а також текст опису котів як хижих тварин, з великою кількістю орфографічних та синтаксичних помилок.

Cats is similar in anatomy, to the others field with a strong flexible body quick reflexes, sharp, retractable claws. And tith adapted – to killing small prey.

3.2.2 Проведення експерименту

Розпочнемо експеримент записом цих текстів у чотири окремі текстові файли. Далі запускаємо програму і отримуємо перші результати які будуть наведені нижче.

Треба зазначити що на даному етапі програма парсер не може розпізнавати файли один за іншим та не має черги завдань для виконання, а також вона не містить графічного інтерфейсу. Це означає що кожний файл буде розпізнаватись окремо. У майбутньому планується надати можливість інтегрувати програму парсер у вигляді бібліотеки до проектів на Java, а також планується підтримка менеджера залежностей Maven.

Запуск програми пасрера буде виконуватися у середовищі розробки IntelliJ IDEA (рис. 3.1).

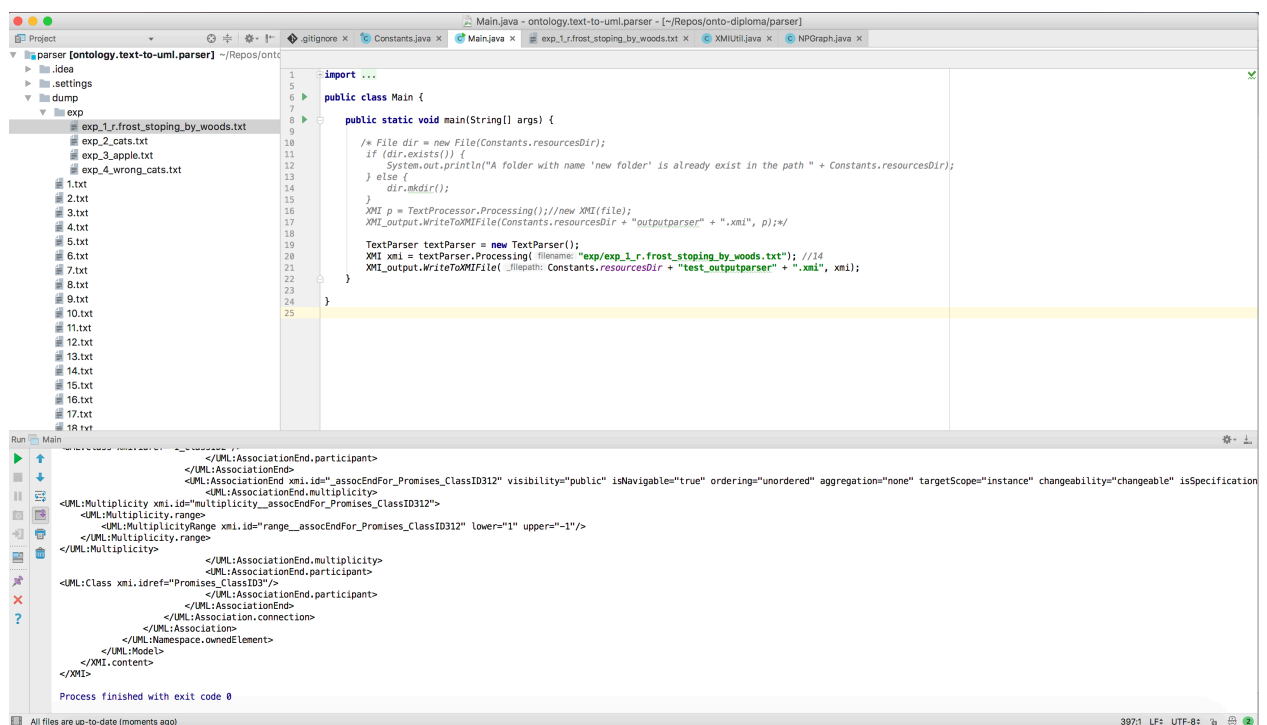


Рисунок 3.1 – Запуск програми у середовищі для Java розробки IntelliJ IDEA

3.2.3 Отримання вихідних даних

Першим був отриманий результат для куплету з віршу [9].

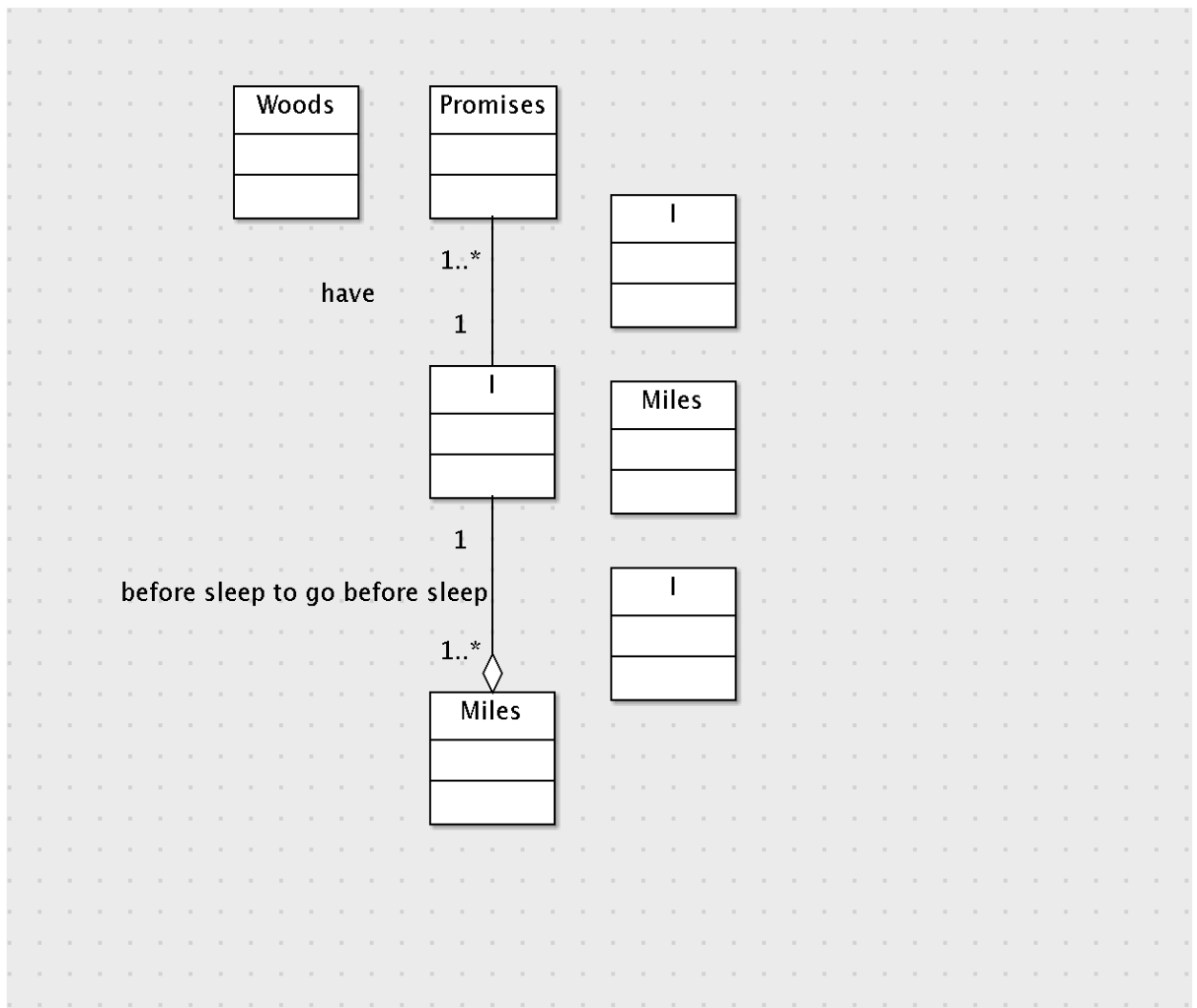


Рисунок 3.2 – UML діаграма для куплету з віршу [9]

Другим був отриманий результат для короткого опису котів як хижих тварин, одразу зазначемо що це є найкращий результат на сьогодні. Цей текст використовувався як тестовий при розробці програми парсера.

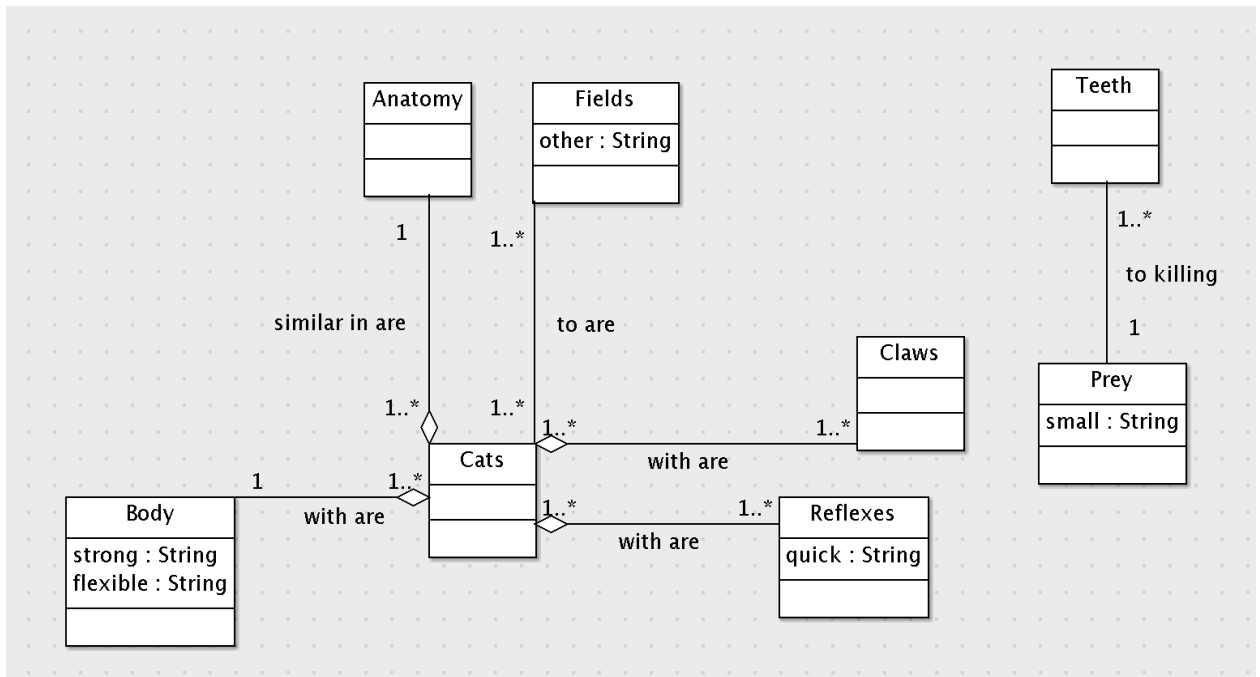


Рисунок 3.3 – UML діаграма отримана з тексту про котів як хижих тварин

Третім був отриманий результат для тексту про заснування компанії Apple. Цей результат вже виглядає неправильним та заплутаним. Також неможливо розмістити його повністю на холсті програми ArgoUML.

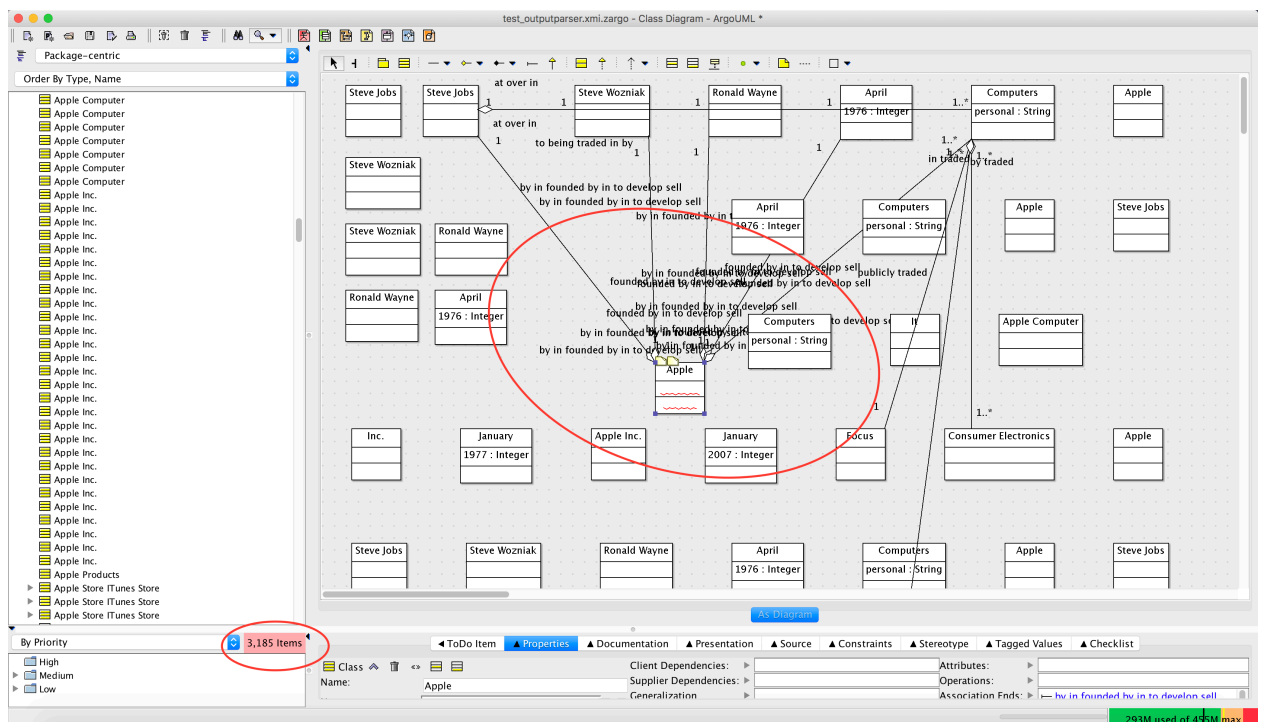


Рисунок 3.4 – UML діаграма з тексту про компанію Apple

Розглянемо результат для тексту про котів як хижих тварин з орфографічними та синтаксичними помилками.

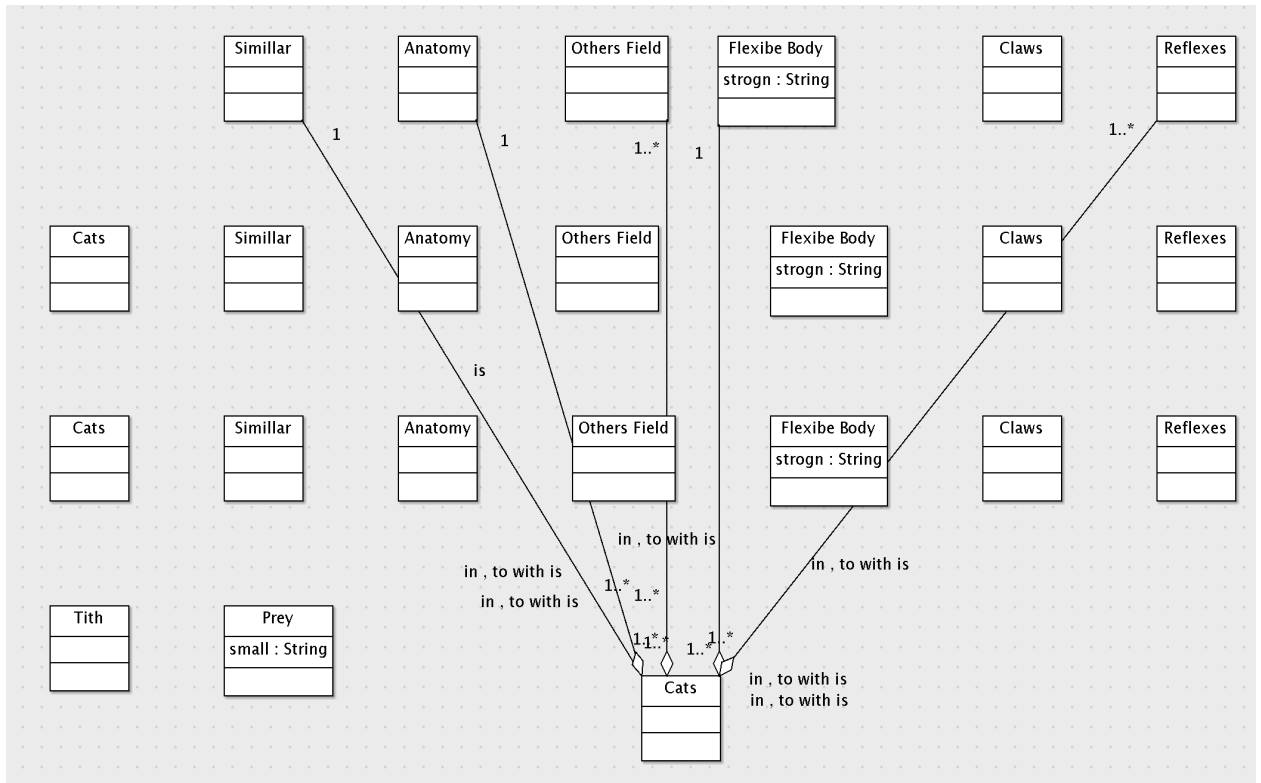


Рисунок 3.5 – UML діаграма опису котів як хижих тварин отримана з тексту який містить орфографічні та синтаксичні помилки

Останнім розглянемо результати парсингу семантично насичених текстів.

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

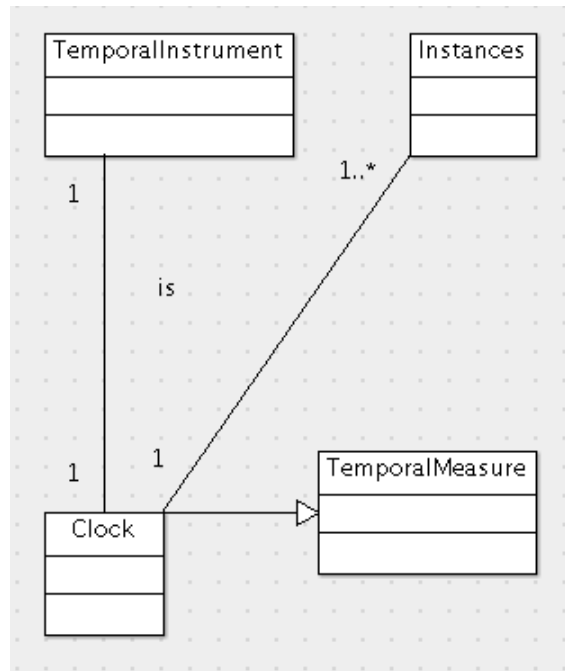


Рисунок 3.6 – UML діаграма визначення Clock (короткий текст)

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).

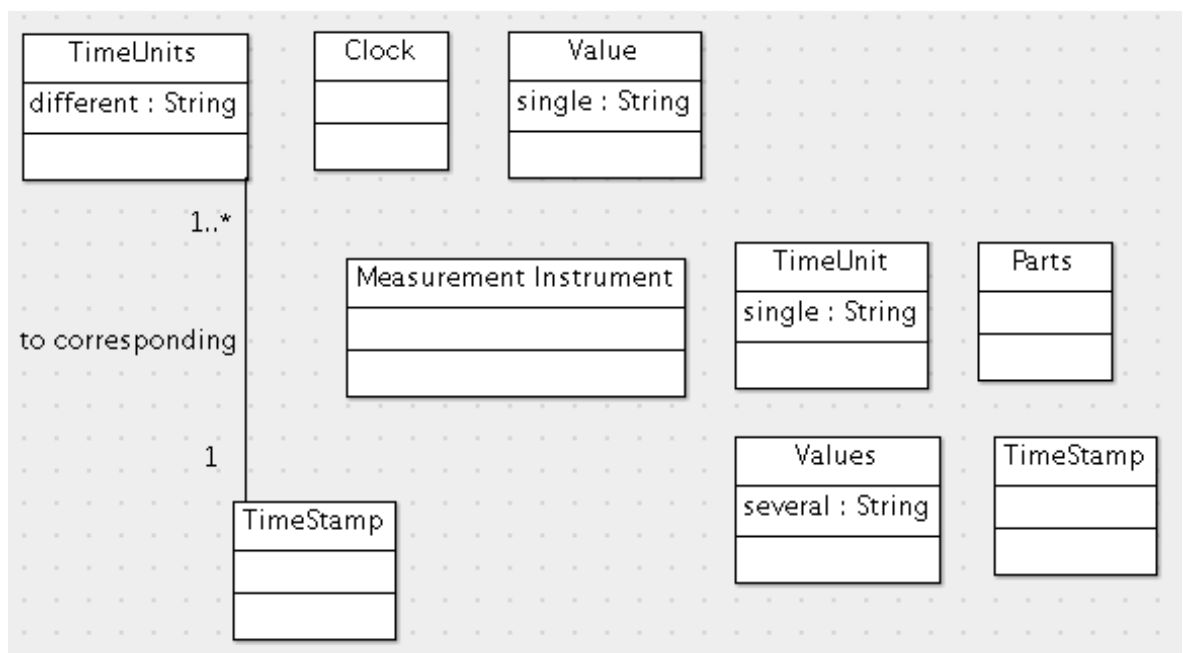


Рисунок 3.7 – UML діаграма визначення Clock (довгий текст)

3.3 Перевірка конвертації текстових сутностей в UML елементи

Конвертація текстових сутностей у UML елементи, за правилами з таблиця. 2.2 для перших двох випадків, пройшла більш менш успішно. Це можна оцінити по тому, як були конвертовані іменники, дієслова та інші частини мови, але в останніх двох випадках рисунок 3.4 та рисунок 3.5 не можна з точністю сказати що всі слова були конвертовані правильно, також з'явилась велика кількість дублікатів у UML діаграмах, що вказує на недосконалість використаного алгоритму конвертації природного тексту в UML (рис. 3.4). А також вказує на недосконалість механізму парсингу текста бібліотеки Stanford Core NLP яка не може пропускати, або ж автоматично виправляти слова та знаки пунктуації.

3.3.1 Перевірка коректності конвертації іменників (підметів) в класи

Мабуть цю задачу програма парсер виконала найуспішніше, практично усі іменники конвертовані у правильні UML сутності (класи), про що свідчать UML діаграми на рисунках 3.2-3.5. Для діаграм на рисунках 3.4-3.5 парсер намагається створити UML класи з іменників навіть якщо результати завідомо некоректні.

Також треба зазначити про отримання досить добрих результатів парсингу семантично насичених текстів (рис. 3.6), здається це найкращий результат, але тільки якщо використовувати невеликі об'єми тексту (рис. 3.7), інакше результат також може бути непередбачуваним.

3.3.2 Перевірка коректності конвертації дієслів (присудків) в залежності

З цим завданням для UML діаграм на рисунках 3.2-3.3 програма парсер впоралась вдало. Але для діаграм на рисунках 3.4-3.5 неможливо точно назвати процент успішності виконання завдання.

Діаграма на рисунку 3.4 почала будуватись коли парсер Core NLP видав більше ніж 3000 залежностей між частинами мови, більшість з яких на мою думку виявляється неправильними семантично. Це є причиною чому ці залежності (в основному дієслова) були інтерпретовані неправильно.

Діаграма на рисунку 3.5 збудована базуючись на завідомо некоректному прикладі тексту, але алгоритм побудови зв'язків між дієсловами та іменниками намагається коректно інтерпретувати навіть такий завідомо неправильний результат парсингу бібліотеки Core NLP.

Також є проблема яка пов'язана з конвертацією дієслів тільки як зв'язків між класами (іменниками). Можна також уявити дієслово як метод класу, наприклад слово stop чудово підходить у якості метода для класу Horse. У майбутньому якщо цей проект буде розвинутий, треба обов'язково надати таку можливість та удосконалити алгоритм конвертації у UML.

3.3.3 Перевірка коректності конвертації прикметника в атрибути класу

Прикметники, як частини мови, конвертуються у UML властивості класів доволі вдало, але тут є одна проблема з недосконалістю правил конвертації у таблиця. 2.2 а відповідно і з алгоритмом конвертації у UML. Це проблема з типами властивостей наприклад: String, Int, Double, Bool. Можна вирішити це для простих та загальноприйнятих типів даних в Java. Але це дуже проблематично, якщо програма парсер буде намагатись створити власні

типи даних. Також дуже проблематично отримати коректну інформацію від Core NLP, в такому виді який би дозволяв задати власний тип даних для прикметника.

Альтернативним вирішенням цієї проблеми, міг би бути підхід, коли частина іменників була б конвертована у UML інтерфейси, наприклад такі прикметники як quick, strong, small чудово виглядали б як інтерфейси. А клас Cat (рис. 3.3) міг би їх імплементувати.

3.3.4 Перевірка коректності конвертації відносин один до багатьох, багато до багатьох та інших

Як можна побачити на рисунках 3.2-3.5 конвертація зв'язків один до одного, один до багатьох, багато до багатьох, переважно виконана правильно. Це результат того що, для визначення конкретного типу зв'язку, в основному, використовується множина слова. Core NLP дуже непогано впоралась з наданням інформації про однину або множину, що у свою чергу сприяло простоті, та коректності роботи алгоритму, у частині конвертації таких типів зв'язків.

3.3.5 Перевірка коректності конвертації типів відносин (association, generalization, aggregation)

Цю задачу, там де це можливо, а також там де було надано достатньо інформації про такі типи зв'язків, програма парсер виконала добре (рис. 3.2-3.3). Але залишається дуже багато питань до правил конвертації таблиця. 2.2 у частині надання зв'язкам конкретного типу. Наступними діями, по удосконаленню програми парсера, повинні стати дії з розширення поточних правил у цій частині.

3.4 Аналіз та оцінка проведеного експерименту

Аналізуючи отримані результати, можна дійти висновку, що програма парсер виконує базові функції конвертації тексту природною мовою в UML діаграми згідно правил у таблиця. 2.2, але успішність конвертації залежить від багатьох факторів. Два найважливіші з них це: довжина тексту та його коректність з точки зору орфографії та семантики. Третім найважливішим фактором може бути коректність самих правил конвертації таблиця 2.1 і таблиця 2.2.

3.4.1 Оцінка отриманих результатів

Результати у вигляді діаграм на рисунках 3.2-3.5 доводять що, програма парсер не є стабільною на данному етапі і може надавати для подальшого аналізу та редагування, як добре сформовані діаграми, так і абсолютно неприродні та тяжкі для розуміння.

Так як планувалося що ці діаграми будуть надані для редагування та аналізу, з подальшою конвертацією у мову OWL в проєкті [2], то такі результати як на рисунках 3.4-3.5 буде неможливо конвертувати правильно, що робить цю програму парсер як найслабкішою ланкою, так і найважливішою у ланцюжку створення онтологій з природного тексту.

3.4.2 Визначення вузьких місць і можливості їх оптимізації

Мабуть найслабкішим місцем у програмному продукті є те що, на сьогодні не визначені повні базові правила для конвертації природної мови в UML, такі правила могли б міститись не на одній сторінці тексту, та могли б зайняти не один місяць часу на їх визначення. На мою думку наступним

кроком розвитку даного ПЗ, повинно бути, намагання чітко визначити хоча б базові правила конвертації для кожної частини мови.

Наступною проблемою є те що, цей проект дуже залежить від бібліотеки Core NLP та її результатів роботи, які є не ідеальними. Вирішенням цієї проблеми була б доробка та удосконалення бібліотеки разом з співтовариством відкритого коду яке постійно проводить різні вдосконалення, та надає нову функціональність.

Також на сьогодні, ця програма не має інтерфейсу і навіть не може гнучко працювати у консольному режимі. Це необхідно виправити, але це також є найменшою проблемою з перерахованих.

3.5 Висновки

Проведення експерименту є одною з найважливіших частин роботи дипломного проекту. Адже це дозволяє нам визначити переваги та недоліки поточних результатів та подальший напрямок розвитку розробленого програмного забезпечення.

Результатом проведення експерименту є:

- а) створення різних як позитивних так і негативних умов для роботи програмного забезпечення;
- б) тестування розробленого програмного забезпечення моделюючи різні події розвитку;
- в) аналіз і оцінка роботи окремого модулю;
- г) визначення вузьких місць та можливостей їх покращення.

Отримані результати насамперед будуть використані для планування наступних етапів розробки та вдосконалення даного програмного забезпечення.

ВИСНОВКИ

У першій частині цієї роботи, було розглянуто основні принципи обробки текста природною мовою за допомогою засобів програмного забезпечення. Був наданий короткий опис найбільш часто досліджуваних завдань у NLP. Були наведені основні напрямки використання, а також були описані поточні види та підходи до семантичного аналізу тексту. Наступними у цій частині були розглянуті засоби зберігання та візуального уявлення тексту такі як XMI, XML та UML, але найголовнішою частиною цього розділу є опис бібліотеки Stanford Core NLP. Ця бібліотека є найголовнішим модулем ПЗ для парсингу текста природною мовою та конвертацією у UML.

Друга частина роботи є практичною, в цій частині було розглянуто специфіку та особливості реалізації програмного продукту, механізм побудови UML діаграм та створення правил конвертації природної мови, роботу з форматом XMI, а також програмний механізм реалізації ПЗ. Практичні завдання які були поставлені у цій частині, були успішно виконані у розділі програмної реалізації. Що ж стосується правил конвертації у UML то можна зазначити наступне, треба ще достатньо часу на дослідження для того щоб створити універсальні правила для такого типу конвертацій. Це може бути темою ще для багатьох кваліфікаційних робіт та наукових паперів.

У третій частині цієї роботи, надані результати експериментальних досліджень для парсингу текста природною мовою та його перетворення на UML діаграму. У результаті декількох експериментів були визначені як сильні так і слабкі сторони програмного продукту, а також була доведена працездатність даного ПЗ у реальних умовах використання. На підставі цих результатів можна стверджувати що програма надає коректні результати роботи тільки у випадку якщо вхідні тексти не містять більш ніж два або три речення.

Підсумовуючи проведену роботу, можна стверджувати, що основна мета була досягнута, але це ніяк не означає що отриманий результат є ідеальним. Наступним у подальшому розвитку даного ПЗ буде, розширення правил конвертації тексту у UML діаграму, удосконалення та оптимізація алгоритму перетворення тексту у UML, а також розширення бібліотеки Stanford Core NLP.

ПЕРЕЛІК ПОСИЛАНЬ

1. Тарануха В.Ю. Інтелектуальна обробка текстів, 2014 [Електронний ресурс] Режим доступу: <http://cyb.univ.kiev.ua/library/books/taranukha-40.pdf>
2. Василейко О.В. Розробка програмного забезпечення для графічного аналізу відповідності онтологій вимогам [Електронний ресурс] Режим доступу: <https://github.com/alexnodejs/onto-argouml>
3. Natural language processing, [Електронний ресурс] Режим доступу: https://en.wikipedia.org/wiki/Natural_language_processing
4. Stanford CoreNLP – Core natural language software [Електронний ресурс] Режим доступу: <https://stanfordnlp.github.io/CoreNLP/>
5. XML Metadata Interchange [Електронний ресурс] Режим доступу: https://en.wikipedia.org/wiki/XML_Metadata_Interchange
6. XML [Електронний ресурс] Режим доступу: <https://www.w3.org/XML/>
7. Unified Modeling Language [Електронний ресурс] Режим доступу: https://uk.wikipedia.org/wiki/Unified_Modeling_Language
8. Stanford Parser [Електронний ресурс] Режим доступу: <http://nlp.stanford.edu:8080/parser/index.jsp>
9. Robert Frost, Stopping by Woods on a Snowy Evening [Електронний ресурс] Режим доступу: <https://www.poetryfoundation.org/poems-and-poets/poems/detail/42891>
10. Java Stanford NLP: Part of Speech labels [Електронний ресурс] Режим доступу: <https://stackoverflow.com/questions/1833252/java-stanford-nlp-part-of-speech-labels/1833718#1833718>
11. Farid Meziane, Nikos Athanasakis, Sophia Ananiadou, Generating Natural Language Specifications From UML Class Diagrams [Електронний ресурс] Режим доступу: <http://usir.salford.ac.uk/1670/1/RE-Meziane-Final.pdf>

12. Marie-Catherine de Marneffe and Christopher, D. Manning, Stanford typed dependencies manual, September 2008, [Электронный ресурс] Режим доступа: https://nlp.stanford.edu/software/dependencies_manual.pdf
13. ArgoUML User Manual : A tutorial and reference description by Alejandro Ramirez, Philippe Vanpeperstraete, Andreas Rueckert, Kunle Odutola, Jeremy Bennett, Linus Tolke, and Michiel van der Wulp
14. Eugene Alforov Text to UML [Электронный ресурс] Режим доступа: <https://gitlab.com/alforov/text-to-uml>
15. Java Code to XML scheme [Электронный ресурс] Режим доступа: <https://www.ibm.com/developerworks/java/tutorials/j-jibx1/>
16. Ermolayev V., Keberle N., Harth A.: Part I: Refining Temporal Representations using OntoElect. In: Extended Semantic Web Conference, 2016 Heraklion, Crete, May 29, 2016
17. Ermolayev V.: Gravitation and Fitness in Ontology Dynamics. In: KIT, AIFB: Kolloquium Angewandte Informatik Jan. 26, 2016, Karlsruhe, Germany
18. Vadim Ermolayev V.: Toward a Syndicated Ontology of Time for the Semantic. In: KIT, AIFB February, 2016, Karlsruhe, Germany
19. OpenNLP Tutorial [Электронный ресурс] Режим доступа: <http://www.programcreek.com/2012/05/opennlp-tutorial/>
20. Stanford POS tagger in Eclipse [Электронный ресурс] Режим доступа: <https://rajvardhan.wordpress.com/2012/11/11/stanford-pos-tagger-in-eclipse/>
21. How to use Opennlp to do part-of-speech tagging [Электронный ресурс] Режим доступа: <http://blog.pengyifan.com/how-to-use-opennlp-to-do-part-of-speech-tagging/>
22. Temporal Representations, [Электронный ресурс] Режим доступа: http://km.aifb.kit.edu/sites/qs-sdm/wiki/Temporal_Representations
23. Frequently Asked Questions [Электронный ресурс] Режим доступа: <https://stanfordnlp.github.io/CoreNLP/faq.html#corenlp-runs-out-of-memory>

24. Stanford JavaNLP API Documentation [Электронный ресурс]
Режим доступа: <https://nlp.stanford.edu/nlp/javadoc/javanlp/overview-summary.html>
25. Universal Dependencies v2 [Электронный ресурс] Режим доступа:
<http://universaldependencies.org/en/dep/>
26. A Rule-based Part-of-Speech and Morphological Tagging Toolkit
[Электронный ресурс] Режим доступа: <http://rdrpostagger.sourceforge.net/>
27. Working XML: UML, XMI, and code generation, Part 2
[Электронный ресурс] Режим доступа:
<https://www.ibm.com/developerworks/library/x-wxxm24/>
28. Общие механизмы XMI [Электронный ресурс] Режим доступа:
http://book.uml3.ru/sec_1_8
29. Onto Diploma [Электронный ресурс] Режим доступа:
<https://github.com/alexnodejs/onto-diploma>

Додаток А

- | | | |
|-----|------|---|
| 1. | CC | Coordinating conjunction (Координаційний сполучник) |
| 2. | FW | Foreign word (Іноземні слова) |
| 3. | IN | Preposition or subordinating conjunction (Прийменник або підрядний сполучник) |
| 4. | JJ | Adjective (Прикметник) |
| 5. | CD | Cardinal number (Кардинальне число) |
| 6. | NN | Noun (Іменник) |
| 7. | NNP | Proper noun, singular (Власні назви, однина) |
| 8. | PRP | Personal pronoun (Особисті займенники) |
| 9. | NNS | Noun, plural (Іменник, множина) |
| 10. | PRP | Personal pronoun (Особовий займенник) |
| 11. | RB | Adverb (Прислівник) |
| 12. | SYM | Symbol (Символ) |
| 13. | UH | Interjection (Вигук) |
| 14. | VB | Verb, base form (Дієслово) |
| 15. | VBP | Verb, non3rd person singular present (Дієслово без третьої особи однини теперішнього форми) |
| 16. | VCN | Verb, past participle (Дієслово, дієприкметник минулого часу) |
| 17. | VBG | Verb, gerund or present participle (Дієслово, герундій або дієприкметник теперішнього часу) |
| 18. | TO | to (Частка to) |
| 19. | VBZ | Verb, 3rd person singular present (Дієслово, третя особа однини теперішнього часу) |
| 20. | ADVP | Adverb phrase (Прислівникова фраза) |
| 21. | ADJP | Adjective phrase (Прикметникова фраза) |
| 22. | PP | Prepositional phrase (Прийменникова фраза) |

23. SBAR Subordinate clause (Підрядне речення)

Додаток Б

```
public static final String singleNounSet[] = new String[] {"NN",  
"NNP", "PRP"};  
public static final String pluralNounSet[] = new String[] {"NNS"};  
public static final String adjectiveSet[] = new String[] {"JJ", "CD",  
"RB"};  
public static final String verbSet[] = new String[] {"VBP", "VBN",  
"VBG", "IN", "TO", "VBZ", "ADVP", "VB"};  
public static final String joinVerbSet[] = new String[] {"ADJP", "PP",  
"SBAR"};  
public static final String conjVerbSet[] = new String[] {"CC", ",", ""};  
public static final String aggregationSet[] = new String[] {"IN"};  
public static final String generalizationSet[] = new String[] {"IN"};  
public static final String generalizationSetWords[] = new String[]  
{"of"};
```

Додаток В

```

<UML:Association xmi.id="_AssociationID8" name=""
isSpecification="false" isRoot="false" isLeaf="false"
isAbstract="false">
  <UML:Association.connection>
    <UML:AssociationEnd xmi.id="_assocEndFor_Horse_ClassID08"
visibility="public" isNavigable="true" ordering="unordered"
aggregation="aggregate" targetScope="instance"
changeability="changeable" name="think queer To stop without near
Between of" isSpecification="false" isRoot="false" isLeaf="false"
isAbstract="false">
      <UML:AssociationEnd.multiplicity>
        <UML:Multiplicity
xmi.id="multiplicity__assocEndFor_Horse_ClassID08">
          <UML:Multiplicity.range>
            <UML:MultiplicityRange
xmi.id="range__assocEndFor_Horse_ClassID08" lower="1" upper="1"/>
          </UML:Multiplicity.range>
        </UML:Multiplicity>
      </UML:AssociationEnd.multiplicity>
      <UML:AssociationEnd.participant>
        <UML:Class xmi.idref="Horse_ClassID0"/>
      </UML:AssociationEnd.participant>
    </UML:AssociationEnd>
    <UML:AssociationEnd xmi.id="_assocEndFor_It_ClassID28"
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_It_ClassID28">
          <UML:Multiplicity.range>
            <UML:MultiplicityRange
xmi.id="range__assocEndFor_It_ClassID28" lower="1" upper="1"/>
          </UML:Multiplicity.range>
        </UML:Multiplicity>
      </UML:AssociationEnd.multiplicity>
      <UML:AssociationEnd.participant>
        <UML:Class xmi.idref="It_ClassID2"/>
      </UML:AssociationEnd.participant>
    </UML:AssociationEnd>
  </UML:Association.connection>
</UML:Association>

```

Додаток Г

```

public static void getNPTrees(Tree tree, List<Tree> nodesNP) {
    List<Tree> subtrees = tree.getChildrenAsList();
    for (Tree subtree : subtrees) {
        if (BaseTreeUtil.isNP(subtree)
            && ! BaseTreeUtil.isHasPhrasesNP(subtree)) {
            nodesNP.add(subtree);
        }

        getNPTrees(subtree, nodesNP);
    }
}

public static void getRelatedNP(
    Tree tree,
    Tree nodeNP,
    Tree root,
    List<NodeTreeData> connectedNodesNP) {
    List<Tree> children = tree.getChildrenAsList();
    for (Tree child : children) {

        if (child.equals(nodeNP)) {
            . . .

            if (!VPUtil.hasConjVP(treeVP)
                && isVP(treeVP.firstChild())) {
                . . .

                for (Tree vpTree : parentVPS) {
                    List<NodeTreeData> connectedNodes =
                        new ArrayList<NodeTreeData>();
                    NPUtil.getNPwithPath(
                        vpTree,
                        connectedNodes,
                        root,
                        vpTree);

                    connectedNodesNP.addAll(connectedNodes);
                }
            } else {
                . . .
            }
        }

        getRelatedNP(child, nodeNP, root, connectedNodesNP);
    }
}

```

Додаток Д

Назва анотатора	Опис
Tokenize	“Токенізує” текст. А також зберігає характер зміщення кожного маркера в полі введення тексту.
Clean Xml	Видаляє XML лексеми з документа.
Sent. split	Розділяє послідовність лексем і створює речення.
POS	Читає текст на деякій мові і визначає частини мови такі як іменник, дієслово, прикметник.
Lemma	Формує слово-леми.
NER	Визначає іменовані (Персона, Місце розташування, Організацію), чисельні (Гроші, Число, Порядок, Відсоток) та часові (Дата, Час, Тривалість) об’єкти.
RegexNER	Метою даного анотатора є створення простого фреймворку для роботи з регулярними виразами. Наприклад, список регулярних виразів для ідеології, національності, релігії та власних назв.
Sentiment	Реалізує модель настроїв (Socher et al’s sentiment model). Підключає бінарне дерево речень до CoreMap компонента
Truecase	Виявляє істинний порядок та значення для лексем в тексті, де ця інформація була втрачена, наприклад, якщо весь текст був написаний у верхньому регістрі.
Parse	Забезпечує повний синтаксичний аналіз, використовуючи залежності та похідні
Dep. Parse	Забезпечує швидкий синтаксичний аналіз.
Coref.	Релізує займенниковий аналіз (anaphora resolution).

Relation	Знаходить відносини між двома суб'єктами.
Nat. Log	Визначає сферу квантифікатора та маркер полярності. Наприклад, для речення «всі кішки мають хвости», анотатор визначить “всі” як квантифікатор з предметної сферою [1, 2) і об'єктом сферою [2, 4).
Quote	Виявляє лапки в тексті